Querying UML Class Diagrams

Andrea Calì[2,3], Georg Gottlob[1,3,4], Giorgio Orsi[1,4], and Andreas Pieris[1]

[1] Department of Computer Science, University of Oxford, UK
[2] Dept. of Computer Science and Inf. Systems, Birkbeck University of London, UK
[3] Oxford-Man Institute of Quantitative Finance, University of Oxford, UK
[4] Institute for the Future of Computing, Oxford Martin School, UK
andrea@dcs.bbk.ac.uk,
{georg.gottlob,giorgio.orsi,andreas.pieris}@cs.ox.ac.uk

Abstract. UML Class Diagrams (UCDs) are the best known class-based formalism for conceptual modeling. They are used by software engineers to model the intensional structure of a system in terms of classes, attributes and operations, and to express constraints that must hold for every instance of the system. Reasoning over UCDs is of paramount importance in design, validation, maintenance and system analysis; however, for medium and large software projects, reasoning over UCDs may be impractical. Query answering, in particular, can be used to verify whether a (possibly incomplete) instance of the system modeled by the UCD, i.e., a snapshot, enjoys a certain property. In this work, we study the problem of querying UCD instances, and we relate it to query answering under guarded Datalog$^{\pm}$, that is, a powerful Datalog-based language for ontological modeling. We present an expressive and meaningful class of UCDs, named *Lean UCD*, under which conjunctive query answering is tractable in the size of the instances.

1 Introduction

Developing complex systems requires accurate design and early prototyping. To avoid the cost of fixing errors at later stages of a project, system designers use models of the final system to negotiate the system design, ensure all resulting requirements, and rule out unintended behavior manifesting itself during the system's lifetime. Models are also of paramount importance for software maintenance and for recovering the structure of a legacy and undocumented software.

UML Class Diagrams. The *Unified Modeling Language (UML)*[1] is one of the major tools for the design of complex systems such as software, business processes and even organizational structures. UML models were proposed by the *Object Modeling Group (OMG)* to diagrammatically represent the static and dynamic, e.g., behavioral aspects of a system, as well as the use cases the system will operate in. More specifically, *UML class diagrams (UCDs)* are widely used to represent the classes (types or entities) of a domain of interest, the associations (relationships) among them, their attributes (fields) and operations (methods).

[1] http://www.omg.org/spec/UML/2.4.1/

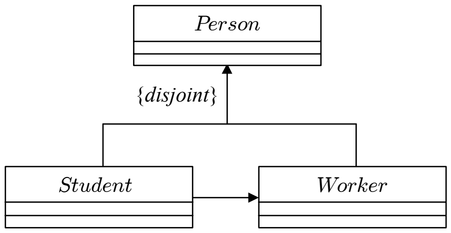

Fig. 1. A UCD which is not fully satisfiable

UCDs for complex projects become very large. Therefore, reasoning tasks such as verifying that a UCD is satisfiable, meaning that the model is realizable, can easily become unfeasible in practice. As a consequence, it is critical to adopt automated procedures to reason on the diagrams and to ensure that the final system will behave as planned.

When a UCD models a software system, the typical reasoning tasks include the following (see [7,37] for additional reasoning tasks):

1. Checking for *satisfiability* (or *consistency*) of the UCD, i.e., checking whether the class diagram admits at least one instantiation, that is, an instance of the system modeled by the UCD that satisfies the diagram.
2. Checking for *full satisfiability* (or *strong consistency*) of the UCD, i.e., checking whether the class diagram admits at least one instantiation where all classes and associations are non-empty. For example, consider the UCD $\mathcal{G}$ of Figure 1, which expresses that each student is a worker, and also that students and workers are disjoint sets of persons. It is easy to see that $\mathcal{G}$ is satisfiable, but not fully satisfiable since the class *Student* must be empty.
3. *Querying* the UCD, i.e., verifying whether a given property — expressed as a query — holds for a given instance of the system modeled by the UCD. This is the reasoning task that we address in this work.

(Full) Satisfiability of UCDs. The seminal work by Berardi et al. [7] established that reasoning on UCDs is hard. In fact, even satisfiability of UCDs is EXPTIME-complete w.r.t. the size of the given diagram. The EXPTIME-hardness is obtained by a reduction from the problem of checking the satisfiability of a concept in $\mathcal{ALC}$ KBs [23,26]. The EXPTIME membership is obtained by providing a polynomial translation of UCD constructs to $\mathcal{DLR}_{ifd}$ KBs [16].

The above results have been refined and extended to full satisfiability of UCDs by Artale et al. [2] and Kaneiwa et al. [30]; upper (resp., lower) bounds are obtained by a reduction to (resp., from) satisfiability of UCDs. In [2], classes of UCDs were identified for which statisfiability is NP-complete and NLOGSPACE-complete by restricting the constructs allowed in the diagrams. In [30], it has also been shown that there exists a fragment of UCDs for which full satisfiability is in PTIME. Moreover, in the same paper, a fragment of UCDs has been identified that allows to check satisfiability in constant time, since the expressive power of its constructs is not enough to capture any unsatisfiable UCD.

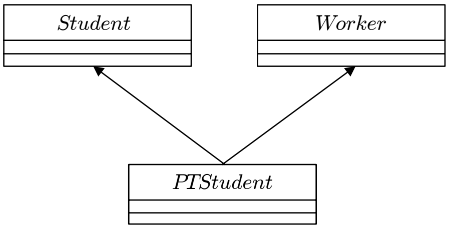

Fig. 2. The UCD of Example 1

Despite its expressiveness, the language of UCDs is often insufficient to express all constraints the designer would like to enforce. For this reason, the OMG consortium devised the *Object Constraint Language (OCL)*[2], which allows for expressing arbitrary constraints on UCDs. OCL is a powerful language, but is not widely adopted due to its complex syntax and ambiguous semantics.

Example 1. Consider the UCD shown in Figure 2 which represents the fact that part-time students are, at the same time, workers and regular students. However, the fact that whenever a student is also a worker, then necessarily (s)he must be a part-time student, cannot be expressed using UCDs. We can express such a constraint using the FOL expression

$$\forall X\ \mathit{Student}(X) \land \mathit{Worker}(X) \rightarrow \mathit{PTStudent}(X).$$

Note that the FOL constraint above corresponds to the OCL expression

```
context PTStudent inv:
  Student.allInstances -> forAll ( s: Student |
    Worker.allInstances -> forAll ( w: Worker |
      s=w implies c.oclIsTypeOf(PTStudent)
    )
  )
```

For OCL syntax and semantics we refer the interested reader to [39]. ■

Reasoning on UML with OCL constraints is considerably harder. It is well known that satisfiability of UCDs extended with OCL constraints is undecidable since it amounts to checking satisfiability of arbitrary first-order formulas. Following Queralt et al. [36,38], the approaches can be classified into three families:

1. Unrestricted OCL constraints without guaranteeing termination, except for specific cases [24,37].
2. Unrestricted OCL constraints with terminating, but incomplete reasoning procedures [8,9,28,40].
3. Restricted classes of OCL constraints with both terminating and complete reasoning procedures [38].

[2] `http://www.omg.org/spec/OCL/2.3/`

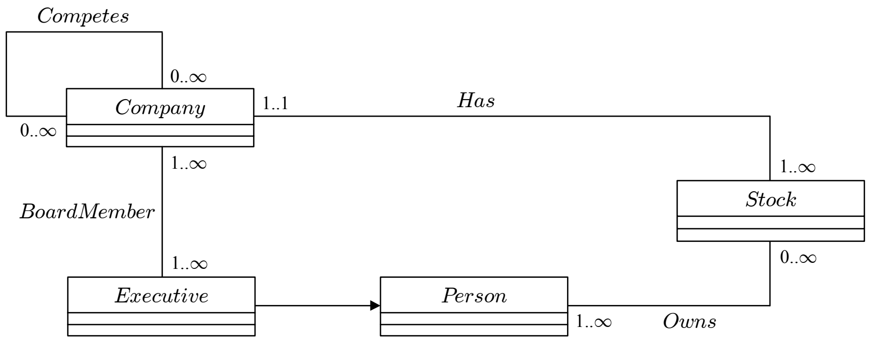

Fig. 3. UCD for the trading scenario

Querying UCDs. UCD satisfiability is an *intensional* property as it depends only on the class diagram and the OCL constraints, without involving any instance of the system modeled by the UCD. On the other hand, in many cases it is useful to reason over instances together with the diagram. For example, a typical task in specification recovery [20] is the reconstruction of a model of an unknown system involving low-level information obtained directly from a running instance of the system. The analyst usually starts from a partial specification (i.e., a UCD) and refines it based on information provided by instances, e.g., by verifying that they are consistent with the specification, and by adjusting the specification when they are not. However, instance-data collected during the analysis is partial but poorly structured. As a consequence they can easily become very large, emphasizing the importance of having procedures that are capable of handling very large instances. In this setting, query answering is a very useful tool for checking whether a property, not expressible diagrammatically, holds. More formally, given an instance D of a system modeled by a UCD $\mathcal{G}$, we can verify whether a property, represented as a query q, holds by checking whether q is a logical consequence of D and $\mathcal{G}$. This problem is known, in the knowledge representation (resp., database) community, as *ontological query answering* (resp., *query answering over incomplete databases*).

Example 2. Consider the UCD of Figure 3 representing a simplified high-frequency trading system. The diagram models companies and the associated stocks together with board members and stakeholders. The *conjunctive query*

$$\begin{aligned}\mathit{Conflict} \leftarrow\ & \mathit{Person}(P), \mathit{Company}(C_1), \mathit{Company}(C_2), \mathit{Stock}(S),\\ & \mathit{BoardMember}(P, C_1), \mathit{Owns}(P, S), \mathit{Has}(C_2, S),\\ & \mathit{Competes}(C_1, C_2)\end{aligned}$$

can be used to detect whether the system allows persons to be, at the same time, in the board of a company and owners of shares of a competing company, and are therefore operating in a conflict of interest. ■

Conjunctive query answering under UCDs and OCL constraints is undecidable in its general form, and has been addressed mainly by reducing it to answering queries in known formalisms such as Prolog [21,41]. Decidable fragments of UCDs have been identified by comparing their expressive power with that of known description logics [3,14,18]. By leveraging on the results of Berardi et al. [7], Calvanese et al. [15] and Lutz [33], it is easy to see that the *combined complexity* of conjunctive query answering over UCDs (without OCL constraints), that is, the complexity w.r.t. the combined size of the the query, the system instance, and the corresponding diagram, is decidable and EXPTIME-complete; this is shown by a reduction from conjunctive query answering under $\mathcal{ALC}$ KBs, and a reduction to conjunctive query answering under $\mathcal{DLR}_{f,id}$ KBs. Calvanese et al. [18] showed the coNP-completeness w.r.t. *data complexity*, i.e., the complexity calculated by considering only the system instance as part of the input, while the query and the diagram are considered fixed. Artale et al. [3] established that fragments of UCDs are captured by the description logic $DL\text{-}Lite_{horn}^{(\mathcal{HN})}$, for which query answering is NP-complete w.r.t. combined complexity, and in AC_0 w.r.t. data complexity. This result subsumes the tractability results for restricted classes of UCDs provided by [18].

Contributions. In this work, we study conjunctive query answering over UCDs and (a restricted class of) OCL constraints that are instrumental to the enforcement of specific assumptions that are commonly adopted for UCDs. We relate the problem to query answering under guarded Datalog$^{\pm}$ [11], a powerful Datalog-based ontological language under which query answering is not only decidable, but also tractable w.r.t. data complexity. In particular, we identify an expressive fragment of UCDs with a limited form of OCL constraints, named *Lean UCD*, which translates into guarded Datalog$^{\pm}$, that features tractable conjunctive query answering w.r.t. data complexity.

Roadmap. After providing some preliminary notions in Section 2, we introduce the UML class diagram formalism, and describe its semantics in terms of first-order logic in Section 3. In Section 4, we present Lean UCD, and we study query answering under the proposed formalism. Finally, Section 5 draws some conclusions and delineates future research directions.

2 Theoretical Background

As we shall see, query answering under UCDs can be reduced to query answering under relational constraints, in particular, tuple-generating dependencies. Therefore, in this section, we recall some basics on relational instances, (Boolean) conjunctive queries, tuple-generating dependencies, and the chase procedure relative to such dependencies.

Alphabets. We define the following pairwise disjoint (infinite) sets of symbols: a set Γ of *constants*, that constitute the "normal" domain of a database, a set Γ_N of *labeled nulls*, used as placeholders for unknown values, and thus can be also seen as (globally) existentially-quantified variables, and a set Γ_V of (regular) *variables*, used in queries and dependencies. Different constants represent different values (*unique name assumption*), while different nulls may represent the same value.

A lexicographic order is defined on $\Gamma \cup \Gamma_N$, such that every value in Γ_N follows all those in Γ. We denote by $\mathbf{X}$ sequences (or sets, with a slight abuse of notation) of variables or constants $X_1, \ldots, X_k$, with $k \geqslant 0$. Throughout, let $[n] = \{1, \ldots, n\}$, for any integer $n \geqslant 1$.

Relational Model. A *relational schema* $\mathcal{R}$ (or simply *schema*) is a set of *relational symbols* (or *predicates*), each with its associated arity. A *term* t is a constant, null, or variable. An *atomic formula* (or simply *atom*) has the form $r(t_1, \ldots, t_n)$, where r is an n-ary relation and $t_1, \ldots, t_n$ are terms. Conjunctions of atoms are often identified with the sets of their atoms. A *relational instance* (or simply *instance*) I for a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $r(\mathbf{t})$, where r is an n-ary predicate of $\mathcal{R}$ and $\mathbf{t} \in (\Gamma \cup \Gamma_N)^n$. A *database* is a finite relational instance.

Substitutions and Homomorphisms. A *substitution* from one set of symbols S_1 to another set of symbols S_2 is a function $h : S_1 \to S_2$ defined as follows: $\varnothing$ is a substitution (empty substitution), and if h is a substitution, then $h \cup \{X \to Y\}$ is a substitution, where $X \in S_1$ and $Y \in S_2$. If $X \to Y \in h$, then we write $h(X) = Y$. A *homomorphism* from a set of atoms A_1 to a set of atoms A_2 is a substitution h from the set of terms of A_1 to the set of terms of A_2 such that: if $t \in \Gamma$, then $h(t) = t$, and if $r(t_1, \ldots, t_n)$ is in A_1, then $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n))$ is in A_2.

(Boolean) Conjunctive Queries. A *conjunctive query (CQ)* q of arity n over a schema $\mathcal{R}$, written as q/n, is an assertion the form $q(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over $\mathcal{R}$, and q is an n-ary predicate that does not occur in $\mathcal{R}$. $\varphi(\mathbf{X}, \mathbf{Y})$ is called the *body* of q, denoted as *body*(q). A *Boolean conjunctive query (BCQ)* is a CQ of arity zero. The *answer* to a CQ q/n over an instance I, denoted as $q(I)$, is the set of all n-tuples $\mathbf{t} \in \Gamma^n$ for which there exists a homomorphism $h : \mathbf{X} \cup \mathbf{Y} \to \Gamma \cup \Gamma_N$ such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$ and $h(\mathbf{X}) = \mathbf{t}$. A BCQ has only the empty tuple $\langle\rangle$ as possible answer, in which case it is said that has positive answer. Formally, a BCQ has *positive* answer over I, denoted as $I \models q$, iff $\langle\rangle \in q(I)$, or, equivalently, $q(I) \neq \varnothing$.

Tuple-Generating Dependencies. A *tuple-generating dependency (TGD)* σ over a schema $\mathcal{R}$ is a first-order formula $\forall \mathbf{X} \forall \mathbf{Y}\, \varphi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of σ, and denoted as *body*(σ) and *head*(σ), respectively. Such σ is satisfied by an instance I for $\mathcal{R}$, written as $I \models \sigma$, iff, whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$, then there exists an extension h' of h, i.e., $h' \supseteq h$, such that $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq I$. We write $I \not\models \sigma$ if I violates σ. Given a set of TGDs Σ, we say that I satisfies Σ, denoted as $I \models \Sigma$, iff I satisfies all the TGDs of Σ. Conversely, we say that I violates Σ, written as $I \not\models \Sigma$, iff I violates at least one TGD of Σ.

Query Answering under TGDs. Given a database D for a schema $\mathcal{R}$, and a set of TGDs Σ over $\mathcal{R}$, the answers we consider are those that are true in *all* models of D w.r.t. Σ, i.e., all instances that contain D and satisfy Σ. Formally, the *models* of D w.r.t. Σ, denoted as $mods(D, \Sigma)$, is the set of all instances I such that $I \models D \cup \Sigma$. The *answer* to a CQ q/n w.r.t. D and Σ, denoted as $ans(q, D, \Sigma)$, is the set of n-tuples $\{\mathbf{t} \mid \mathbf{t} \in q(I)$, for each $I \in mods(D, \Sigma)\}$. The answer to a BCQ q w.r.t. D and Σ is *positive*, denoted as $D \cup \Sigma \models q$, iff $\langle\rangle \in ans(q, D, \Sigma)$, or, equivalently, $ans(q, D, \Sigma) \neq \varnothing$.

Given a CQ q/n over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, a set Σ of TGDs over $\mathcal{R}$, and an n-tuple $\mathbf{t} \in \Gamma^n$, CQAns is defined as the problem whether $\mathbf{t} \in ans(q, D, \Sigma)$. In case that q is a BCQ (and thus, $\mathbf{t}$ is the empty tuple $\langle\rangle$), the above problem is called BCQAns. Notice that these two problems under general TGDs are undecidable [6], even when the schema and the set of TGDs are fixed [10], or even when the set of TGDs is restricted to a single rule [4]. Following Vardi's taxonomy [42], the *data complexity* of the above problems is the complexity calculated taking only the database as input, while the query and the set of dependencies are considered fixed. The *combined complexity* is the complexity calculated considering as input, together with the database, also the query and the set of dependencies.

It is well-known that the above decision problems are LOGSPACE-equivalent; this result is implicit in [19], and stated explicitly in [10]. Henceforth, we thus focus only on BCQAns, and all complexity results carry over to CQAns.

The TGD Chase Procedure. The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [34], and later for checking query containment [29]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted instance satisfies the dependencies. By abuse of terminology, we shall use the term "chase" interchangeably for both the procedure and its result. The building block of the chase procedure is the so-called *TGD chase rule.*

Definition 1 (TGD Chase Rule). *Consider an instance I for a schema $\mathcal{R}$, and a TGD $\sigma : \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$. If σ is* applicable *to I, i.e., there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$, but there is no extension h' of h (i.e., $h' \supseteq h$) that maps $\psi(\mathbf{X}, \mathbf{Z})$ to I, then: (i) define $h' \supseteq h$ such that $h'(Z_i) = z_i$, for each $Z_i \in \mathbf{Z}$, where $z_i \in \Gamma_N$ is a "fresh" labeled null not introduced before, and following lexicographically all those introduced so far, and (ii) add to I the set of atoms in $h'(\psi(\mathbf{X}, \mathbf{Z}))$, if not already in I.*

Given a database D and a set Σ of TGDs, the chase algorithm for D w.r.t. Σ consists of an exhaustive application of the TGD chase rule, which leads to a (possibly infinite) instance denoted as $chase(D, \Sigma)$. We assume that the chase algorithm is *fair*, i.e., each TGD that must be applied during the construction of $chase(D, \Sigma)$ is eventually applied.

Example 3. Consider the set Σ constituted by the TGDs $\sigma_1 : \forall X, Y\, r(X, Y) \wedge s(Y) \rightarrow \exists Z\, r(Z, X)$ and $\sigma_2 : \forall X, Y\, r(X, Y) \rightarrow s(X)$. Let D be the database $\{r(a, b), s(b)\}$. During the chase of D w.r.t. Σ, we first apply σ_1 and we add the atom $r(z_1, a)$, where $z_1 \in \Gamma_N$. Also, σ_2 is applicable and we add the atom $s(a)$. Now, σ_1 is applicable and the atom $r(z_2, z_1)$ is obtained, where $z_2 \in \Gamma_N$.

Then, σ_2 is applicable and the atom $s(z_1)$ is generated. It is straightforward to see that there is no finite chase. Satisfying both TGDs σ_1 and σ_2 would require to built the infinite instance $\{r(a,b), s(b), r(z_1,a), s(a), r(z_2,z_1), s(z_1), r(z_3,z_2), s(z_2), \ldots\}$, where, for each $i > 0$, $z_i \in \Gamma_N$. ■

The fact that the chase algorithm is fair allows us to show that chase of a database D w.r.t. a set of TGDs Σ is a *universal model* of D w.r.t. Σ, i.e., for each $I \in \mathit{mods}(D,\Sigma)$, there exists a homomorphism from $\mathit{chase}(D,\Sigma)$ to I [22,25]. Using this fact it can be shown that the chase is a useful tool for query answering under TGDs. More precisely, the problem whether the answer to a BCQ q is positive w.r.t. a database D and a set of TGDs Σ, is equivalent to the problem whether q is entailed by the chase of D w.r.t. Σ.

Theorem 1 ([22,25]). *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. $D \cup \Sigma \models q$ iff $\mathit{chase}(D,\Sigma) \models q$.*

Guarded Datalog$^{\pm}$. Since query answering under TGDs is undecidable, several classes of TGDs have been proposed under which the problem becomes decidable, and even tractable w.r.t. data complexity (see, e.g., [4,25,31]). In particular, Datalog$^{\pm}$ [12] is a family of languages for ontological modeling based on classes of TGDs under which query answering is decidable and, in almost all cases, tractable w.r.t. data complexity. Datalog$^{\pm}$ proved to be a valid alternative to description logics in many database and knowledge management applications.

A member of the Datalog$^{\pm}$ family which is of special interest for our work is *guarded Datalog*$^{\pm}$ [10,11]. A TGD σ is *guarded* if it has a body-atom which contains all the universally quantified variables of σ. Such atom is called the *guard atom* (or simply *guard*) of σ. The non-guard atoms are the *side atoms* of σ. For example, the TGD $r(X,Y), s(Y,X,Z) \rightarrow \exists W s(Z,X,W)$ is guarded (via the guard $s(Y,X,Z)$), while the TGD $r(X,Y), r(Y,Z) \rightarrow r(X,Z)$ is not guarded. Note that sets of guarded TGDs (with single-atom heads) are theories in the *guarded fragment* of first-order logic [1].

As shown in [10], the chase constructed under a set of guarded TGDs has finite treewidth, which, intuitively speaking, means that the chase is a tree-like structure. This is exactly the reason why query answering under guarded TGDs is decidable. The data and combined complexity of query answering under guarded TGDs have been investigated in [11] and [10], respectively.

Theorem 2 ([10,11]). BCQAns *under guarded TGDs is* PTIME*-complete w.r.t. data complexity,* EXPTIME*-complete in the case of bounded arity, and 2*EXPTIME*-complete w.r.t. combined complexity.*

3 UML Class Diagrams

As already mentioned in Section 1, UML class diagrams (UCDs) describe the static structure of a system by showing the system's classes, their attributes and operations, and the relationships among the classes. In this section, we describe

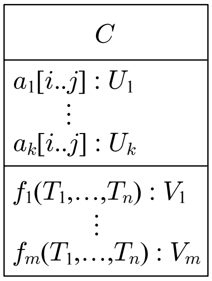

Fig. 4. Class representation

the semantics of each construct of UCDs in terms of first-order logic (FOL) generalized by counting quantifiers. The formalization adopted in this paper is based on the one presented in [30].

Classes. A *class* is graphically represented as shown in Figure 4, i.e., as a rectangle divided into three parts. The top part contains the *name* of the class which is unique in the diagram, the middle part contains the *attributes* of the class, and the bottom part contains the *operations* of the class, that is, the operations associated to the instances of the class. Note that both the middle and the bottom part are optional.

An *attribute assertion* of the form $a[i..j] : T$ states that the class C has an attribute a of *type*[3] T, where the optional multiplicity $[i..j]$ specifies that a associates to each instance of C at least i and at most j instances of T. When there is no lower (resp., upper) bound on the multiplicity, the symbol 0 (resp., ∞) is used for i (resp., j). Notice that attributes are unique within a class. However, different classes may have attributes with the same name, possibly with different types.

An operation of a class C is a function from the instances of C, and possibly additional parameters, to objects and values. An *operation assertion* of the form $f(T_1, \ldots, T_n) : T$ asserts that the class C has an operation f with $n \geqslant 0$ parameters, where its i-th parameter is of type T_i and its result is of type T. Let us clarify that the class diagram represents only the *signature*, that is, the name of the functions as well as the number and the types of their parameters, and the type of their result. The actual behavior of the function, which is not part of the diagram, can be represented using OCL constraints. Notice that operations are unique within a class. However, different classes may have operations with the same name, possibly with different signature, providing that they have the same number of parameters.

We are now ready to give the formal translation of a UML class definition into FOL. A class C is represented by a FOL unary predicate C. An attribute a for class C corresponds to a binary predicate a, and the attribute assertion

[3] For simplicity, types, i.e., collections of values such as integers, are considered as classes, i.e., as collections of objects.

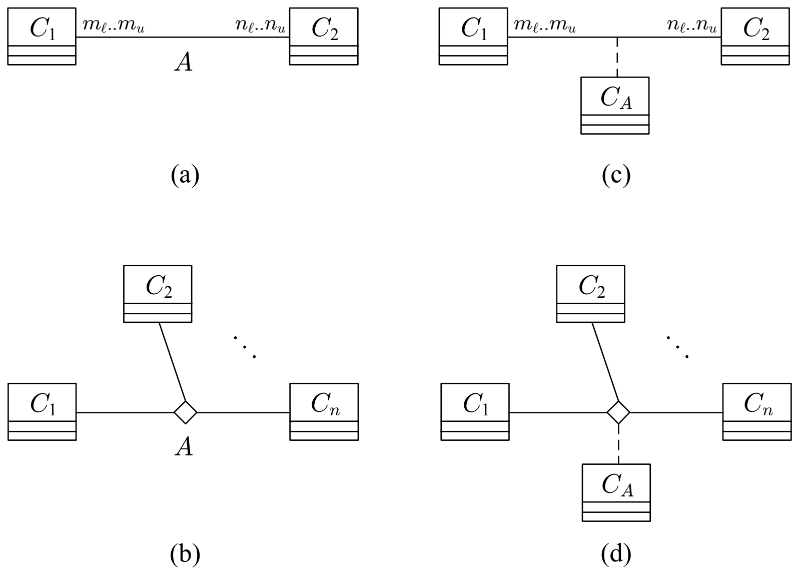

Fig. 5. Association representation

$a[i..j] : T$ is translated into two FOL assertions. The first one asserts that for each instance c of class C, an object c' related to c by the attribute a is an instance of T:

$$\forall X, Y\ C(X) \wedge a(X,Y) \rightarrow T(Y).$$

The second one asserts that for each instance c of class C, there exist at least i and at most j different objects related to c by a:

$$\forall X\ C(X) \rightarrow \exists_{\geqslant i} Z\, a(X,Z) \wedge \exists_{\leqslant j} Z\, a(X,Z).$$

If $i = 0$, which implies that there is no lower bound, then the corresponding FOL assertion is of the form $\forall X\ C(X) \rightarrow \exists_{\leqslant j} Z\ a(X,Z)$. Dually, if $j = \infty$, which implies that there is no upper bound, then the obtained assertion is of the form $\forall X\ C(X) \rightarrow \exists_{\geqslant i} Z\ a(X,Z)$.

An operation f, with $m \geqslant 0$ parameters, for class C corresponds to an $(m+2)$-ary predicate f, and the operation assertion $f(T_1,\ldots,T_m) : T$ is translated into the FOL assertions

$$\forall X, Y_1,\ldots,Y_m, Z\ C(X) \wedge f(X,Y_1,\ldots,Y_m,Z) \rightarrow \textstyle\bigwedge_{i=1}^{m} T_i(Y_i) \wedge T(Z),$$

which imposes the correct typing for the parameters and the result, and

$$\begin{aligned}\forall X, Y_1,\ldots,Y_m, Z_1, Z_2\ C(X) &\wedge f(X,Y_1,\ldots,Y_m,Z_1)\\ &\wedge f(X,Y_1,\ldots,Y_m,Z_2) \rightarrow Z_1 = Z_2,\end{aligned}$$

i.e., the operation f is a function from the instances of C and the parameters to the result.

Associations. An *association* is a relation between the instances of two or more classes, that are said to *participate* in the association. Names of associations are unique in the diagram. A binary association A between two classes C_1 and C_2 is graphically represented as in Figure 5a. The multiplicity $n_\ell..n_u$ specifies that each instance of class C_1 can participate at least n_ℓ times and at most n_u times to A; similarly we have the multiplicity $m_\ell..m_u$ for C_2.

Clearly, we can have also n-ary associations which relate several classes as shown in Figure 5. As already discussed in [7], while multiplicity constraints in binary associations appear natural, for non-binary associations they do not correspond to an intuitive property of the multiplicity. Due to this fact, their presence in non-binary associations is awkward to a designer, and also they express a constraint which is (in general) too weak in practice. Therefore, in this paper, multiplicity in non-binary associations it is assumed to be always $0..\infty$. Notice that in [30] arbitrary multiplicity constraints in non-binary associations are allowed.

An association can have an *association class* which describes properties of the association such as attributes and operations. A binary association between C_1 and C_2 with an association class C_A is graphically represented as in Figure 5c. An n-ary association can also have an association class as depicted in Figure 5d.

Let us now give the formal translation of an association definition into FOL. An n-ary association A corresponds to an n-ary predicate A. Assuming that A is among classes $C_1, \ldots, C_n$, A is translated into the FOL assertion

$$\forall X_1, \ldots, X_n \; A(X_1, \ldots, X_n) \rightarrow \textstyle\bigwedge_{i=1}^{n} C_i(X_i).$$

If A has a related association class C_A, then we have also the FOL assertions (in the sequel, r_n is an $(n+1)$-ary auxiliary predicate)

$$\forall X_1, \ldots, X_n, Y \; A(X_1, \ldots, X_n) \wedge r_n(X_1, \ldots, X_n, Y) \rightarrow C_A(Y),$$

which types the association A,

$$\forall X_1, \ldots, X_n \; A(X_1, \ldots, X_n) \rightarrow \exists Z \; r_n(X_1, \ldots, X_n, Z),$$

i.e., for each instance $\langle x_1, \ldots, x_n \rangle$ of A, there exists at least one object related to $\langle x_1, \ldots, x_n \rangle$ by r_n,

$$\begin{aligned} \forall X_1, \ldots, X_n, Y_1, Y_2 \; A(X_1, \ldots, X_n) &\wedge r_n(X_1, \ldots, X_n, Y_1) \\ &\wedge r_n(X_1, \ldots, X_n, Y_2) \rightarrow Y_1 = Y_2, \end{aligned}$$

that is, for each instance $\langle x_1, \ldots, x_n \rangle$ of A, there exists at most one object related to $\langle x_1, \ldots, x_n \rangle$ by r_n, and

$$\begin{aligned} \forall X_1, \ldots, X_n, Y_1, \ldots, Y_n, Z \; r_n(X_1, \ldots, X_n, Z) &\wedge r_n(Y_1, \ldots, Y_n, Z) \\ &\wedge C_A(Z) \rightarrow \textstyle\bigwedge_{i=1}^{n} X_i = Y_i, \end{aligned}$$

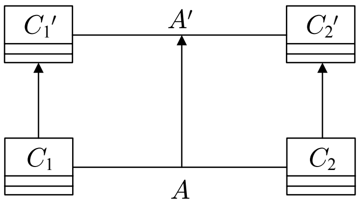

Fig. 6. Association generalization

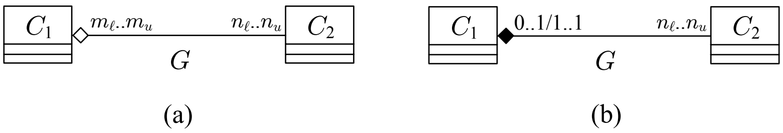

Fig. 7. Aggregation and composition

which imposes that there are no two different instances of C_A that are related by r_n with the same instance of A; this rule guarantees the faithful representation of the association A by C_A, according to the original UML semantics.

Now, for a binary association A among C_1 and C_2 with multiplicities $m_\ell..m_u$ and $n_\ell..n_u$, we have also the FOL assertions

$$\begin{array}{l}\forall X\ C_1(X) \rightarrow \exists_{\geqslant n_\ell} Z\ A(X,Z) \wedge \exists_{\leqslant n_u} Z\ A(X,Z),\\ \forall X\ C_2(X) \rightarrow \exists_{\geqslant m_\ell} Z\ A(Z,X) \wedge \exists_{\leqslant m_u} Z\ A(Z,X).\end{array}$$

We can also have *association generalization* such that an n-ary association A' between $C_1', \ldots, C_n'$ generalizes the n-ary association A between $C_1, \ldots, C_n$ (see Figure 6 for the binary case). This feature is captured by the FOL assertion

$$\forall X_1, \ldots, X_n\ A(X_1, \ldots, X_n) \rightarrow A'(X_1, \ldots, X_n).$$

A special kind of binary associations are *aggregations* and *compositions*, representing two different forms of *whole-part* or *part-of* relationship. An aggregation (see Figure 7a) between two classes C_1 and C_2 specifies that each instance of C_2, called the *contained class*, is conceptually part of an instance of C_1, called the *container class*; for example, a handle is part of a door. A composition (see Figure 7b) is more specific than aggregation. Composition has a strong *life cycle dependency* between instances of the container class and instances of the contained class. In particular, if the container is destroyed, then each instance that it contains is destroyed as well. Notice that the contained class of a composition must have a multiplicity of 0..1 or 1..1. Clearly, the life cycle dependency of a composition must be considered during the implementation phase of a system, however it is not relevant for query answering purposes. The translation of an aggregation and a composition into FOL is the same as the one given above for

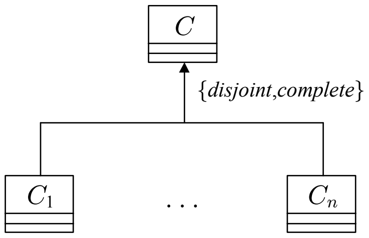

Fig. 8. A class hierarchy

binary associations without an association class (since aggregations and compositions have no association class).

Class Hierarchies. Similar to association generalization, one can use *class generalization* to assert that each instance of a child class is also an instance of the parent class. Several generalizations can be grouped together to form a class hierarchy, as shown in Figure 8. *Disjointness* and *completeness* constraints can also be enforced on a class hierarchy (graphically, by adding the labels {*disjoint*} and {*complete*}). A class hierarchy, as the one in Figure 8, is translated into the FOL assertions

$$\forall X\ C_1(X) \rightarrow C(X),$$
$$\vdots$$
$$\forall X\ C_n(X) \rightarrow C(X),$$

i.e., each instance of C_i is also an instance of C,

$$\forall X\ C_i(X) \rightarrow \textstyle\bigwedge_{j=i+1}^{n} \neg C_j(X),$$

for each $i \in [n-1]$, which specify the disjointness constraints, and

$$\forall X\ C(X) \rightarrow \textstyle\bigvee_{i=1}^{n} C_i(X),$$

which specify the completeness constraints.

Sometimes, it is assumed that all classes in the same hierarchy are disjoint. However, we do not enforce this assumption, and we allow two classes to have common instances. When needed, disjointness can be enforced by means of FOL assertions, called *negative constraints*, of the form

$$\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow \bot,$$

where $\varphi(\mathbf{X})$ is a conjunction of atoms, and $\bot$ denotes the truth constant *false*. Moreover, we do not enforce the *most specific class* assumption, stating that objects in a hierarchy must belong to a single most specific class. Therefore, two classes in a hierarchy may have common instances, even though they may not have a common subclass. When needed, the existence of the most specific class can be enforced by means of *multi-linear TGDs* [11] of the form

$$\forall X\ C_1(X) \wedge \ldots \wedge C_n(X) \rightarrow C_{n+1}(X),$$

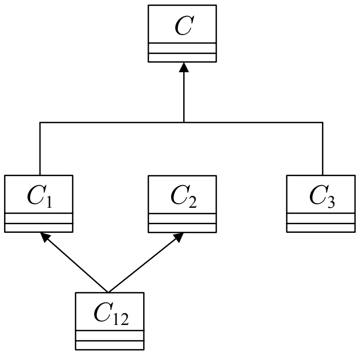

Fig. 9. A class hierarchy with most specific class assumption

where each C_i is a unary predicate representing a class. Observe that multi-linear TGDs are guarded TGDs where each body-atom is a guard. Notice that negative constraints and multi-linear TGDs can be represented using suitable OCL constraints.

For example, besides the assertions representing the hierarchy depicted in Figure 9 (taken from [7]), the most specific class assumption can be expressed by the following FOL assertions:

$$\begin{array}{l} \forall X\ C_1(X) \wedge C_3(X) \rightarrow \bot \\ \forall X\ C_2(X) \wedge C_3(X) \rightarrow \bot \\ \forall X\ C_1(X) \wedge C_2(X) \rightarrow C_{12}(X). \end{array}$$

4 Querying Lean UML Class Diagrams

The main goal of the present work is to study the problem of conjunctive query answering under UCDs. In particular, we are interested to identify an expressive fragment of UCDs which can be encoded in Datalog$^{\pm}$ so that chase-like techniques (for the chase algorithm see Section 2) and known complexity results can be employed.

In this section, we propose mild syntactic restrictions on the full version of UCDs, presented in Section 3, in order to get a fragment with the aforementioned desirable properties, called *Lean UCD*. We then study query answering under the proposed formalism. As we shall see, given a Lean UCD $\mathcal{G}$, by applying the translation of UCDs into FOL assertions given in the previous section — in the following we refer to this translation by τ — on $\mathcal{G}$, we get FOL assertions which have one of the following forms:

1. $\forall \mathbf{X} \forall \mathbf{Y}\, \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms, and $\varphi(\mathbf{X}, \mathbf{Y})$ has an atom that contains all the universally quantified variables; recall that an assertion of this form is called guarded TGD.

2. $\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow X_i = X_j$, where $\varphi(\mathbf{X})$ is a conjunction of atoms, and both X_i and X_j are variables of $\mathbf{X}$; assertions of this form are known as *equality-generating dependencies (EGDs)*.
3. $\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow \perp$, where $\varphi(\mathbf{X})$ is a conjunction of atoms, and $\perp$ denotes the truth constant *false*; recall that assertions of this form are called negative constraints.

The rest of this section is organized as follows. In Subsection 4.1, we formalize Lean UCDs by applying a series of syntactic restrictions on UCDs. Then, the problem of query answering under Lean UCDs is defined and studied in Subsection 4.2.

4.1 Formalizing Lean UCDs

Lean UCDs are obtained by restricting the multiplicity of attributes and binary associations, and by omitting completeness constraints. Formally, a UML class diagram $\mathcal{G}$ is a Lean UCD if the following conditions hold:

1. For each attribute assertion of the form $a[i..j] : T$ of $\mathcal{G}$:
   - $i \in \{0, 1\}$ and $j \in \{1, \infty\}$.
2. For each binary association A of $\mathcal{G}$, either with or without an association class (consider the binary associations depicted in Figures 5a and 5c):
   - $n_\ell, m_\ell \in \{0, 1\}$ and $n_u, m_u \in \{1, \infty\}$,
   - if A generalizes some other binary association of $\mathcal{G}$, then $n_u = m_u = \infty$.
3. There are no completeness constraints in $\mathcal{G}$.

Example 4. The Lean UCD of Figure 10 describes members of a university department working in research groups. In particular, the class hierarchy specifies that students and professors, which are disjoint classes, are members of the department. A member of the department works in at least one research group, and at least one departmental member works in a research group. Moreover, a professor leads at most one research group, while a research group is led by exactly one professor; notice that a professor works in the same group that (s)he leads. Finally, a publication is authored by at least one member of the department. ■

Interestingly, the FOL assertions which represent multiplicities of attributes or multiplicities of binary associations, obtained by applying the translation τ on a Lean UCD $\mathcal{G}$, are guarded TGDs and EGDs. More precisely, from a class C with an attribute assertion $a[i..j] : T$ we get FOL assertions of the form

$$\forall X\ C(X) \rightarrow \exists_{\geqslant 1} Z\ a(X,Z) \equiv \forall X\ C(X) \rightarrow \exists Z\ a(X,Z),$$
$$\forall X\ C(X) \rightarrow \exists_{\leqslant 1} Z\ a(X,Z) \equiv \forall X,Y,Z\ C(X) \wedge a(X,Y) \wedge a(X,Z) \rightarrow Y = Z.$$

From a binary association A among C_1 and C_2 with multiplicities $m_\ell..m_u$ and $n_\ell..n_u$, we get FOL assertions of the form

$$\forall X\ C_1(X) \rightarrow \exists_{\geqslant 1} Z\ A(X,Z) \equiv \forall X\ C_1(X) \rightarrow \exists Z\ A(X,Z),$$
$$\forall X\ C_1(X) \rightarrow \exists_{\leqslant 1} Z\ A(X,Z) \equiv \forall X,Y,Z\ C_1(X) \wedge A(X,Y) \wedge A(X,Z) \rightarrow Y = Z,$$
$$\forall X\ C_2(X) \rightarrow \exists_{\geqslant 1} Z\ A(Z,X) \equiv \forall X\ C_2(X) \rightarrow \exists Z\ A(Z,X),$$
$$\forall X\ C_2(X) \rightarrow \exists_{\leqslant 1} Z\ A(Z,X) \equiv \forall X,Y,Z\ C_2(X) \wedge A(Y,X) \wedge A(Z,X) \rightarrow Y = Z.$$

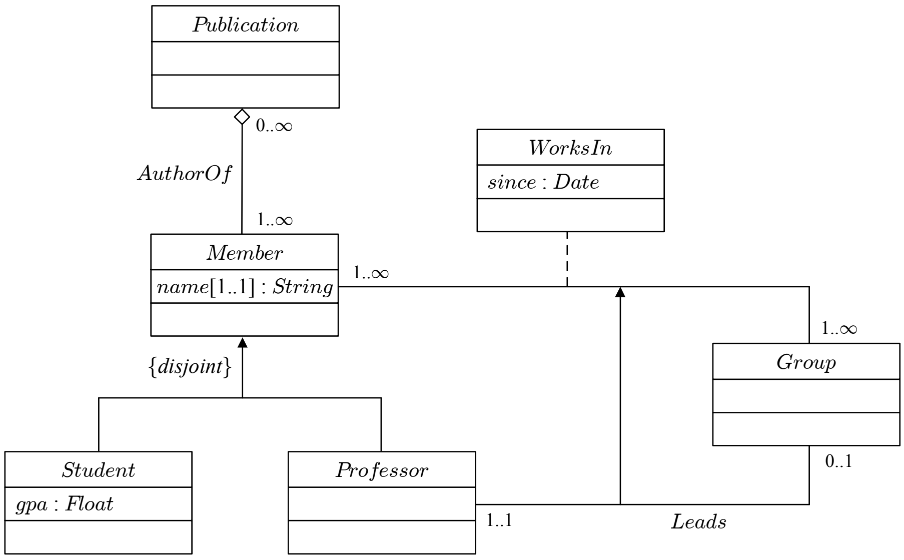

Fig. 10. Lean UCD of Example 4

It is an easy task to verify that, given a Lean UCD $\mathcal{G}$, the set of FOL assertions $\tau(\mathcal{G})$ is constituted by guarded TGDs, EGDs and negative constraints.

4.2 Query Answering under Lean UCDs

The problem of query answering under TGDs (discussed in Section 2) can be naturally extended to sets of TGDs, EGDs and negative constraints. An EGD $\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow X_i = X_j$ is satisfied by a relational instance I if, whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X})) \subseteq I$, then $h(X_i) = h(X_j)$. A negative constraint $\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow \bot$ is satisfied by I if there is no homomorphism h such that $h(\varphi(\mathbf{X})) \subseteq I$.

From the above discussion we immediately get that query answering under Lean UCDs is a well-defined problem. Formally, given a Lean UCD $\mathcal{G}$, a BCQ q (which represents a desirable property of the system; recall Example 2), and a relational database D (which is an instance of the system), the answer to q w.r.t. D and $\mathcal{G}$ is positive, denoted as $D \cup \mathcal{G} \models q$, iff $\langle\rangle \in \mathit{ans}(q, D, \tau(\mathcal{G}))$, or, equivalently, $\mathit{ans}(q, D, \tau(\mathcal{G})) \neq \varnothing$.

In the rest of the paper, given a Lean UCD $\mathcal{G}$ we denote by $\mathcal{R}_\mathcal{G}$ the relational schema associated to $\mathcal{G}$, i.e., the set of predicates occurring in $\tau(\mathcal{G})$, excluding the auxiliary predicates of the form r_i, where $i \geqslant 2$. Moreover, an instance of the system modeled by $\mathcal{G}$ is considered as a (relational) database for $\mathcal{R}_\mathcal{G}$, while a property to be verified is encoded as a BCQ over $\mathcal{R}_\mathcal{G}$.

Elimination of EGDs and Negative Constraints. Recall that our main algorithmic tool for query answering is the chase algorithm. However, the chase algorithm presented in Section 2 can treat only TGDs[4], but in the set of FOL assertions that we obtain by applying τ on a Lean UCD $\mathcal{G}$ we have also EGDs and negative constraints. Interestingly, as we shall see, providing that the logical theory constituted by the given database and the set of assertions $\tau(\mathcal{G})$ is consistent, we are allowed to eliminate the EGDs and the negative constraints, and proceed only with the TGDs.

Let us first concentrate on the EGDs. Consider a Lean UCD $\mathcal{G}$. For notational convenience, in the rest of this section, let $\tau_1(\mathcal{G})$ be the set of TGDs occurring in $\tau(\mathcal{G})$, $\tau_2(\mathcal{G})$ be the set of EGDs occurring in $\tau(\mathcal{G})$, and $\tau_3(\mathcal{G})$ be the set of negative constraints occurring in $\tau(\mathcal{G})$. The semantic notion of *separability* expresses a controlled interaction of TGDs and EGDs, so that the presence of EGDs does not play any role in query answering, and thus can be ignored [11,13]. Similarly, we can define separable Lean UCDs.

Definition 2 (Separability). *A Lean UCD $\mathcal{G}$ is* separable *if, for every database D for $\mathcal{R}_\mathcal{G}$, either $D \not\models \tau_2(\mathcal{G})$, or $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q$ iff $D \cup \tau_1(\mathcal{G}) \models q$, for every BCQ q over $\mathcal{R}_\mathcal{G}$.*

It is possible to show that *each* Lean UCD $\mathcal{G}$ is separable. Lean UCDs, in fact, enjoy a stronger property than separability: during the construction of the chase of a database D for $\mathcal{R}_\mathcal{G}$ w.r.t. $\tau_1(\mathcal{G})$, it is not possible to violate an EGD of $\tau_2(\mathcal{G})$, and therefore, if D satisfies $\tau_2(\mathcal{G})$, then $\mathit{chase}(D, \tau_1(\mathcal{G}))$ is a universal model of D w.r.t. $\tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G})$ which implies separability of $\mathcal{G}$.

Lemma 1. *Each Lean UCD is separable.*

Proof (sketch). Consider a Lean UCD $\mathcal{G}$, and a database D for $\mathcal{R}_\mathcal{G}$ such that $D \models \tau_2(\mathcal{G})$; if $D \not\models \tau_2(\mathcal{G})$, then the claim holds trivially. We need to show that $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q$ iff $D \cup \tau_1(\mathcal{G}) \models q$, for every BCQ q over $\mathcal{R}_\mathcal{G}$. Observe that each model of the logical theory $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G})$ is also a model of $D \cup \tau_1(\mathcal{G})$. Therefore, if $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q$, then $D \cup \tau_1(\mathcal{G}) \models q$. It remains to establish the other direction. It suffices to show that, for each $i \geqslant 0$, $\mathit{chase}^{[i]}(D, \tau_1(\mathcal{G}))$, i.e., the initial segment of $\mathit{chase}(D, \tau_1(\mathcal{G}))$ obtained starting from D and applying i times the TGD chase rule (see Definition 1), does not violate any of the EGDs of $\tau_2(\mathcal{G})$. We proceed by induction on $i \geqslant 0$.

Base Step. Clearly, $\mathit{chase}^{[0]}(D, \tau_1(\mathcal{G})) = D$, and the claim follows since $D \models \tau_2(\mathcal{G})$.

Inductive Step. Suppose that during the i-th application of the TGD chase rule we apply the TGD $\sigma \in \tau_1(\mathcal{G})$ with homomorphism λ, and the atom $\underline{a}$ is obtained. Notice that each set Σ of guarded TGDs can be rewritten into an equivalent set Σ' of guarded TGDs, where each TGD of Σ' has just one head-atom [10]. Therefore, for technical clarity, we assume w.l.o.g. that each TGD of $\tau_1(\mathcal{G})$ has just one head-atom.

[4] Notice that the chase algorithm can be extended to treat also EGDs (see, e.g., [25]). However, such an extended version of the chase algorithm is not needed for the purposes of the present paper.

Consider an EGD $\eta \in \tau_2(\mathcal{G})$. It suffices to show that there is no homomorphism which maps $\mathit{body}(\eta)$ to $\mathit{chase}^{[i]}(D, \tau_1(\mathcal{G}))$. Suppose that η is of the form $\forall X_1, \ldots, X_n, Y_1, Y_2\ A(X_1, \ldots, X_n) \wedge r_n(X_1, \ldots, X_n, Y_1) \wedge r_n(X_1, \ldots, X_n, Y_2) \rightarrow Y_1 = Y_2$. We show that there is no homomorphism that maps the set of atoms $A = \{r_n(X_1, \ldots, X_n, Y_1), r_n(X_1, \ldots, X_n, Y_2)\} \subset \mathit{body}(\eta)$ to $\mathit{chase}^{[i]}(D, \tau_1(\mathcal{G}))$, and thus there is no homomorphism that maps $\mathit{body}(\eta)$ to $\mathit{chase}^{[i]}(D, \tau_1(\mathcal{G}))$. The critical case is when σ is of the form $\forall X_1, \ldots, X_n\ A(X_1, \ldots, X_n) \rightarrow \exists Z\ r_n(X_1, \ldots, X_n, Z)$. In the sequel, let $\underline{a} = r_n(\mathbf{t})$, and consider an arbitrary atom $\underline{a}' = r_n(\mathbf{t}') \in \mathit{chase}^{[i-1]}(D, \tau_1(\mathcal{G}))$. Towards a contradiction, suppose that there exists a homomorphism that maps A to $\{\underline{a}, \underline{a}'\}$. This implies that there exists an extension of λ that maps $\mathit{head}(\sigma)$ to $\underline{a}' \in \mathit{chase}^{[i-1]}(D, \tau_1(\mathcal{G}))$, and hence σ is not applicable with homomorphism λ. But this contradicts the fact that σ has been applied during the i-th application of the TGD chase rule.

By providing a similar argument, we can establish the same fact for all the forms of EGDs that can appear in $\tau_2(\mathcal{G})$, and the claim follows. $\square$

Let us now focus on the negative constraints. Given a Lean UCD, checking whether $\tau_3(\mathcal{G})$ is satisfied by a database D for $\mathcal{R}_\mathcal{G}$ and the set of assertions $\tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G})$ is tantamount to query answering; this is implicit in [11]. More precisely, for each negative constraint ν of the form $\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow \bot$ of $\tau_3(\mathcal{G})$, we evaluate the BCQ $q_\nu : p \leftarrow \varphi(\mathbf{X})$ over $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G})$. If at least one of such queries answers positively, then the logical theory $D \cup \tau(\mathcal{G})$ is inconsistent, and thus $D \cup \mathcal{G} \models q$, for every BCQ q; otherwise, given a BCQ q, it holds that $D \cup \tau(\mathcal{G}) \models q$ iff $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q$, i.e., we answer q by ignoring the negative constraints. The next lemma follows immediately.

Lemma 2. *Consider a Lean UCD $\mathcal{G}$, a BCQ q over $\mathcal{R}_\mathcal{G}$, and a database D for $\mathcal{R}_\mathcal{G}$. Then, $D \cup \mathcal{G} \models q$ iff (i) $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q$ or (ii) there exists $\nu \in \tau_3(\mathcal{G})$ such that $D \cup \tau_1(\mathcal{G}) \cup \tau_2(\mathcal{G}) \models q_\nu$.*

By combining Lemmas 1 and 2, we immediately get the following useful technical result, which implies that query answering under Lean UCDs can be reduced to query answering under guarded TGDs, and thus chase-like techniques and existing complexity results can be employed.

Corollary 1. *Consider a Lean UCD $\mathcal{G}$, a BCQ q over $\mathcal{R}_\mathcal{G}$, and a database D for $\mathcal{R}_\mathcal{G}$. If $D \models \tau_2(\mathcal{G})$, then $D \cup \mathcal{G} \models q$ iff (i) $D \cup \tau_1(\mathcal{G}) \models q$ or (ii) there exists $\nu \in \tau_3(\mathcal{G})$ such that $D \cup \tau_1(\mathcal{G}) \models q_\nu$.*

Complexity of Query Answering. We are now ready to investigate the complexity of BCQAns under Lean UCDs. Before we proceed further, let us analyze the complexity of the problem of deciding whether a database violates the set of EGDs obtained from a Lean UCD.

Lemma 3. *Consider a Lean UCD $\mathcal{G}$, and a database D for $\mathcal{R}_\mathcal{G}$. The problem of deciding whether $D \not\models \tau_2(\mathcal{G})$ is feasible in* PTIME *if $\mathcal{G}$ is fixed, and in* NP *in general.*

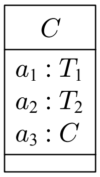

Fig. 11. The construction in the proof of Theorem 3

Proof. It is possible to show that a database D' and a set of BCQs Q can be constructed in PTIME such that $D \not\models \tau_2(\mathcal{G})$ iff $D' \models q$, for some $q \in Q$. The database D' is constructed by adding to D an atom $neq(c_1, c_2)$, for each pair $\langle c_1, c_2 \rangle$ of distinct constants occurring in D, where neq is an auxiliary binary predicate not occurring in $\mathcal{R}_\mathcal{G}$. Clearly, the number of atoms that we need to add in D is at most n^2, where n is the number of constants occurring in D, and thus D' can be constructed in polynomial time. The set Q of BCQs is defined as follows. For each EGD $\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow X_i = X_j$ of $\tau_2(\mathcal{G})$, in Q there exists a BCQ $p \leftarrow \varphi(\mathbf{X}), neq(X_i, X_j)$, where p is a 0-ary auxiliary predicate. Clearly, Q can be constructed in linear time. Clearly, by construction, $D \not\models \tau_2(\mathcal{G})$ iff $D' \models q$, for some $q \in Q$. Since the evaluation of a BCQ over a database is feasible in AC$_0$ if the query is fixed [43], and in NP in general [19], the claim follows. □

We continue to investigate the data complexity of BCQAns under Lean UCDs; recall that the data complexity is calculated by considering only the database as part of the input, while the query and the diagram are considered fixed.

Theorem 3. BCQAns *under Lean UCDs is* PTIME*-complete w.r.t. data complexity.*

Proof. By Corollary 1, given a Lean UCD $\mathcal{G}$, a BCQ q over $\mathcal{R}_\mathcal{G}$, and a database D for $\mathcal{R}_\mathcal{G}$, we can decide whether $D \cup \mathcal{G} \models q$ by applying the following simple algorithm: (1) if $D \not\models \tau_2(\mathcal{G})$, then answer *yes*; (2) if $D \cup \tau_1(\mathcal{G}) \models q$, then answer *yes*; (3) if there exists $\nu \in \tau_3(\mathcal{G})$ such that $D \cup \tau_1(\mathcal{G}) \models q_\nu$, then answer *yes*; (4) answer *no*. Since the diagram is fixed, Lemma 3 implies that the first step can be carried out in PTIME. Recall that $\tau_1(\mathcal{G})$ is a set of guarded TGDs. Since, by Theorem 2, BCQAns under guarded TGDs is feasible in PTIME w.r.t. data complexity, the claimed upper bound follows.

Let us now establish the PTIME-hardness of the problem under consideration. The proof is by reduction from *Path System Accessibility (PSA)* which is PTIME-hard [27]. An instance of PSA is a quadruple $\langle N, E, S, t \rangle$, where N is a set of nodes, $E \subseteq N \times N \times N$ is an accessibility relation (its elements are called *edges*), $S \subseteq N$ is a set of source nodes, and $t \in N$ is a terminal node. The question is whether t is accessible, where a node $v \in N$ is said to be accessible if $v \in S$ or there exist accessible nodes v_1 and v_2 s.t. $\langle v, v_1, v_2 \rangle \in E$.

Let $\mathcal{G}$ be the Lean UCD depicted in Figure 11, which contains also the additional rule $\forall X\ T_1(X) \wedge T_2(X) \rightarrow C(X)$. Clearly, $\tau(\mathcal{G})$ is constituted by

$$\begin{aligned}
&\sigma_1 : \forall X, Y\ C(X) \wedge a_1(X, Y) \rightarrow T_1(Y)\\
&\sigma_2 : \forall X, Y\ C(X) \wedge a_2(X, Y) \rightarrow T_2(Y)\\
&\sigma_3 : \forall X, Y\ C(X) \wedge a_3(X, Y) \rightarrow C(Y)\\
&\sigma_4 : \forall X\ T_1(X) \wedge T_2(X) \rightarrow C(X).
\end{aligned}$$

For the construction of the database D we make use of the nodes and the edges of N and E, respectively, as constants. In particular, the domain of D is the set of constants $\{c_v \mid v \in N\} \cup \{c_e \mid e \in E\}$. For a node $v \in N$, let $e_1^v, \ldots, e_n^v$, where $n \geqslant 0$, be all edges of E that have v as their first component (the order is not relevant). The database D contains, for each node $v \in N$, the following atoms:

– $a_3(c_{e_1^v}, c_v)$ and $a_3(c_{e_{i+1}^v}, c_{e_i^v})$, for each $i \in [n-1]$,
– $a_1(c_u, c_{e_i^v})$ and $a_2(c_w, c_{e_i^v})$, where $e_i^v = \langle v, u, w\rangle$, for each $i \in [n]$.

In addition, D contains an atom $C(c_v)$ for each $v \in S$. Finally, let q be the BCQ $p \leftarrow C(c_t)$. Intuitively speaking, T_1 keeps all the edges $\langle v_1, v_2, v_3\rangle$ where v_2 is accessible, T_2 keeps all the edges $\langle v_1, v_2, v_3\rangle$ where v_3 is accessible, and C keeps all the nodes which are accessible, and also all the edges $\langle v_1, v_2, v_3\rangle$ where v_1 is accessible. Let us say that the above construction is similar to a construction given in [17], where the data complexity of query answering under description logics is investigated. It is easy to verify that $\mathcal{G}$, D and q can be constructed in LOGSPACE. Moreover, observe that only the database D depends on the given instance of PSA, while the diagram and the query are fixed. It remains to show that t is accessible iff $D \cup \mathcal{G} \models q$, or, equivalently, $\mathit{chase}(D, \tau(\mathcal{G})) \models q$. For brevity, in the rest of the proof, a constant c_x, where x is either a node of N or an edge of E, is denoted as x.

($\Rightarrow$) Suppose first that t is accessible. It is possible to show, by induction on the derivation of accessibility, that if a node $v \in N$ is accessible, then $\mathit{chase}(D, \tau(\mathcal{G})) \models q_v$, where q_v is the BCQ $p \leftarrow C(v)$.

Base Step. Suppose that $v \in S$. By construction, $C(v) \in D$, and thus $\mathit{chase}(D, \tau(\mathcal{G})) \models q_v$.

Inductive Step. Clearly, there exists an edge $\langle v, v_1, v_2\rangle \in E$, where both v_1 and v_2 are accessible. By induction hypothesis, both atoms $C(v_1)$ and $C(v_2)$ belong to $\mathit{chase}(D, \tau(\mathcal{G}))$. Let $e_1, \ldots, e_n$ be the edges of E that have v as their first component, where $e_n = \langle v, v_1, v_2\rangle$; assume the same order used in the construction of the database D. Since the atoms $a_1(v_1, e_n)$ and $a_2(v_2, e_n)$ belong to D, and $\mathit{chase}(D, \tau(\mathcal{G})) \models \tau(\mathcal{G})$, the atoms $T_1(e_n)$ and $T_2(e_n)$ belong to $\mathit{chase}(D, \tau(\mathcal{G}))$. Therefore, due to the TGD σ_4, $C(e_n) \in \mathit{chase}(D, \tau(\mathcal{G}))$. Now, by exploiting the atoms $\{a_3(e_{i+1}, e_i)\}_{i \in [n-1]} \subset D$ and the TGD σ_3, it is easy to show that the atom $C(e_1)$ belongs to $\mathit{chase}(D, \tau(\mathcal{G}))$. Finally, since $a_3(e_1, v) \in \mathit{chase}(D, \tau(\mathcal{G}))$, we get that $C(v) \in \mathit{chase}(D, \tau(\mathcal{G}))$. Consequently, $\mathit{chase}(D, \tau(\mathcal{G})) \models q_v$.

($\Leftarrow$) Suppose now that $\mathit{chase}(D, \tau(\mathcal{G})) \models q$. We need to show that t is accessible. We are going to show that, for each $v \in N$, if $C(v) \in \mathit{chase}(D, \tau(\mathcal{G}))$, then v

$$\begin{tikzcd}
C(u_1) \arrow[dr, "\sigma_1"'] & & a_1(u_1,e_n) \arrow[dl, "\sigma_1"] & & C(u_2) \arrow[dr, "\sigma_2"'] & & a_1(u_2,e_n) \arrow[dl, "\sigma_2"] \\
& T_1(e_n) \arrow[drr, "\sigma_4"'] & & & & T_2(e_n) \arrow[dll, "\sigma_4"] & \\
& & & C(e_n) & & a_3(e_n,e_{n\text{-}1}) & \\
& & & \vdots & & \vdots & \\
& & & C(e_2) \arrow[dr, "\sigma_3"'] & & a_3(e_2,e_1) \arrow[dl, "\sigma_3"] & \\
& & & & C(e_1) \arrow[dr, "\sigma_3"'] & & a_3(e_1,v) \arrow[dl, "\sigma_3"] \\
& & & & & C(v) &
\end{tikzcd}$$

Fig. 12. The chase derivation in the proof of Theorem 3

is accessible. Observe that the only way to obtain $C(v)$ during the construction of the chase is as depicted in Figure 12. Since $C(u_1)$ and $C(u_2)$ are database atoms, by construction, both u_1 and u_2 belong to S, and thus they are accessible. Since $e_n = \langle w, u_1, u_2 \rangle$, we immediately get that w is accessible. But observe that $w = v$, and hence v is accessible. This completes the proof. $\square$

We conclude this subsection by investigating the combined complexity of BCQAns under Lean UCDs; recall that the combined complexity is calculated by considering, apart from the database, also the query and the diagram as part of the input. As we shall see, query answering under Lean UCDs can be reduced to query answering under guarded TGDs of bounded arity, and thus (by Theorem 2) we get an EXPTIME upper bound.

Theorem 4. BCQAns *under Lean UCDs is in* EXPTIME *w.r.t. combined complexity.*

Proof. As already discussed in the proof of Theorem 3, given a Lean UCD $\mathcal{G}$, a BCQ q over $\mathcal{R}_{\mathcal{G}}$, and a database D for $\mathcal{R}_{\mathcal{G}}$, we can decide whether $D \cup \mathcal{G} \models q$ by applying the following simple algorithm: (1) if $D \not\models \tau_2(\mathcal{G})$, then answer *yes*; (2) if $D \cup \tau_1(\mathcal{G}) \models q$, then answer *yes*; (3) if there exists $\nu \in \tau_3(\mathcal{G})$ such that $D \cup \tau_1(\mathcal{G}) \models q_\nu$, then answer *yes*; (4) answer *no*. By Lemma 3, we get that the first step can be carried out in NP. Now, in order to show that steps (2) and (3) can be carried out in EXPTIME, we are going to show that the problem of answering a BCQ under the set $\tau_1(\mathcal{G})$ can be reduced to query answering under a set of guarded TGDs of bounded arity, which is in EXPTIME (by Theorem 2).

Any TGD of $\tau_1(\mathcal{G})$ which has one of the following forms is called *harmless*:

$$\begin{aligned}
\forall X_1, \ldots, X_n \; A(X_1, \ldots, X_n) &\rightarrow C_1(X_1), \ldots, C_n(X_n), \\
\forall X_1, \ldots, X_n, Y \; A(X_1, \ldots, X_n) \wedge r'(X_1, \ldots, X_n, Y) &\rightarrow C_A(Y), \\
\forall X_1, \ldots, X_n \; A(X_1, \ldots, X_n) &\rightarrow \exists Z \, r'(X_1, \ldots, X_n, Z), \\
\forall X_1, \ldots, X_n \; A(X_1, \ldots, X_n) &\rightarrow A'(X_1, \ldots, X_n).
\end{aligned}$$

Let Σ_h be the set of harmless TGDs of $\tau_1(\mathcal{G})$. It is not difficult to see that, for every BCQ p over $\mathcal{R}_{\mathcal{G}}$, $D \cup \tau_1(\mathcal{G}) \models p$ iff $\mathit{chase}(\mathit{chase}(D, \Sigma_h)_{\downarrow}, \tau_1(\mathcal{G}) \setminus \Sigma_h) \models p$, where $\mathit{chase}(D, \Sigma_h)_{\downarrow}$ is the (finite) database obtained by *freezing* $\mathit{chase}(D, \Sigma_h)$, i.e., by replacing each null of Γ_N occurring in $\mathit{chase}(D, \Sigma_h)$ with a fresh constant of Γ. Now, observe that atoms of the form $f(c_1, \ldots, c_n)$ are necessarily database atoms since the predicate f does not occur in the head of any of the TGDs of $\tau_1(\mathcal{G})$. Therefore, whenever a TGD of the form $\forall X, Y_1, \ldots, Y_m, Z\ C(X) \wedge f(X, Y_1, \ldots, Y_m, Z) \rightarrow T_1(Y_1), \ldots, T_m(Y_m), T(Z)$ is applied during the chase, the variable X is mapped (via a homomorphism) to a constant. Thus, for each database atom $f(c_1, \ldots, c_{m+2})$, we can replace the above rule with the rule $C(c_1) \rightarrow T_1(c_2), \ldots, T_m(c_{m+1}), T(c_{m+2})$. Let Σ be the set obtained by applying the above transformation on the set $\tau_1(\mathcal{G}) \setminus \Sigma_h$. It holds that, $D \cup \tau_1(\mathcal{G}) \models p$ iff $\mathit{chase}(\mathit{chase}(D, \Sigma_h)_{\downarrow}, \Sigma) \models p$, for every BCQ p over $\mathcal{R}_{\mathcal{G}}$. It is not difficult to see that Σ is a set of guarded TGDs of bounded arity, and the claim follows. □

Notice that the above theorem does not provide tight combined complexity bounds for the problem of query answering under Lean UCDs. The exact bound is currently under investigation, and the results will be presented in an upcoming work. Preliminary findings can be found in an online manuscript[5].

Interestingly, both the data and combined complexity of query answering under Lean UCDs can be reduced by applying further restrictions. In particular, this can be achieved by assuming that each attribute and operation is associated to a unique class, i.e., different classes have disjoint sets of attributes and operations, and also an association A with an association class C_A does not generalize some other association A'. These assumptions allow us to establish the following: given a diagram $\mathcal{G}$, a BCQ q over $\mathcal{R}_{\mathcal{G}}$, and a database D for $\mathcal{R}_{\mathcal{G}}$, there exists a set Σ of multi-linear TGDs over a schema $\mathcal{R}$, where each predicate of $\mathcal{R}$ has bounded arity, such that $D \cup \tau_1(\mathcal{G}) \models q$ iff $D \cup \Sigma \models q$; recall that a multi-linear TGD is a guarded TGD where each body-atom is a guard. Since query answering under multi-linear TGDs is in AC$_0$ w.r.t. data complexity and NP-complete in the case of bounded arity [11], we immediately get that query answering under Lean UCDs that satisfy the above assumptions is in AC$_0$ w.r.t. data complexity and NP-complete w.r.t. combined complexity.

5 Discussion and Future Work

In this work, we have studied the problem of query answering under UML class diagrams (UCDs) by relating it to the problem of query answering under guarded Datalog$^{\pm}$. In particular, we have identified an expressive fragment of UCDs, called Lean UCD, under which query answering is PTIME-complete w.r.t. data complexity and in EXPTIME w.r.t. combined complexity.

In the immediate future we plan to investigate expressive, but still tractable, extensions of Lean UCD with additional constructs such as stratified negation.

[5] `http://dl.dropbox.com/u/3185659/uml.pdf`

Moreover, we are planning to investigate the problem of query answering under Lean UCDs by considering only finite instantiations. This is an interesting and important research direction since, in practice, it is natural (for obvious reasons) to concentrate only on finite instances of a system.

Reasoning on UCDs by considering only finite instantiations has been studied first by Cadoli et al. [9] by building on results of Lenzerini and Nobili [32] established for the Entity-Relationship model. In particular, [9] proposes an encoding of satisfiability of UCDs under finite instantiations into a constraint satisfaction problem, showing that satisfiability of UCDs under finite instantiations is EXPTIME-complete. Later, more efficient algorithms based on linear programming were presented by Maraee and Balaban for specific instances of the problem [35]. Recently, Queralt et al. [38] extended the results of [9] to cope with a restricted form of OCL constraints, called *OCL-Lite*, and shown that satisfiability of UCDs and *OCL-Lite* constraints under finite instantiations is EXPTIME-complete.

An important notion related to query answering under UCDs and OCL constraints is the so-called *finite controllability*. A class of UCDs $\mathcal{C}$ (possibly with OCL constraints) is finitely controllable if the following holds: given a diagram $\mathcal{G}$ that falls in $\mathcal{C}$, an instance D of the system modeled by $\mathcal{G}$, and a query q, $D \cup \mathcal{G} \models q$ iff $D \cup \mathcal{G} \models_{\mathit{fin}} q$, i.e., q is entailed by $D \cup \mathcal{G}$ under arbitrary instantiations iff it is entailed by $D \cup \mathcal{G}$ under finite instantiations only. It is worthwhile to remark that by forbidding functional participation Lean UCD is finitely controllable. This is a consequence of the fact the guarded fragment of first-order logic, and thus guarded Datalog$^{\pm}$, is finitely controllable [5].

Acknowledgements. This research has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement DIADEM no. 246858 and from the Oxford Martin School's grant no. LC0910-019.

References

1. Andréka, H., van Benthem, J., Németi, I.: Modal languages and bounded fragments of predicate logic. J. Philosophical Logic 27, 217–274 (1998)
2. Artale, A., Calvanese, D., Ibáñez-García, Y.A.: Full Satisfiability of UML Class Diagrams. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 317–331. Springer, Heidelberg (2010)
3. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. Artificial Intelligence Res. 36, 1–69 (2009)
4. Baget, J.-F., Leclère, M., Mugnier, M.-L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artif. Intell. 175(9-10), 1620–1654 (2011)
5. Bárány, V., Gottlob, G., Otto, M.: Querying the guarded fragment. In: Proc. of LICS, pp. 1–10 (2010)
6. Beeri, C., Vardi, M.Y.: The Implication Problem for Data Dependencies. In: Even, S., Kariv, O. (eds.) ICALP 1981. LNCS, vol. 115, pp. 73–85. Springer, Heidelberg (1981)
7. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artif. Intell. 168(1-2), 70–118 (2005)

8. Cabot, J., Clariso, R., Riera, D.: Verification of UML/OCL class diagrams using constraint programming. In: Proc. of ICSTW, pp. 73–80 (2008)
9. Cadoli, M., Calvanese, D., De Giacomo, G., Mancini, T.: Finite Model Reasoning on UML Class Diagrams Via Constraint Programming. In: Basili, R., Pazienza, M.T. (eds.) AI*IA 2007. LNCS (LNAI), vol. 4733, pp. 36–47. Springer, Heidelberg (2007)
10. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. In: Proc. of KR, pp. 70–80 (2008); Extended version available from the authors
11. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. In: Proc. of PODS, pp. 77–86 (2009); To appear in the J. of Web Semantics
12. Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., Pieris, A.: Datalog+/-: A family of logical knowledge representation and query languages for new applications. In: Proc. of LICS, pp. 228–242 (2010)
13. Calì, A., Lembo, D., Rosati, R.: On the decidability and complexity of query answering over inconsistent and incomplete databases. In: Proc. of PODS, pp. 260–271 (2003)
14. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. Autom. Reasoning 39(3), 385–429 (2007)
15. Calvanese, D., De Giacomo, G., Lenzerini, M.: On the decidability of query containment under constraints. In: Proc. PODS, pp. 149–158 (1998)
16. Calvanese, D., De Giacomo, G., Lenzerini, M.: Identification constraints and functional dependencies in description logics. In: Proc. of IJCAI, pp. 155–160 (2001)
17. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of KR, pp. 260–270 (2006)
18. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Conceptual Modeling for Data Integration. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 173–197. Springer, Heidelberg (2009)
19. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proc. of STOCS, pp. 77–90 (1977)
20. Chikofsky, E.J., Cross II, J.H.: Reverse engineering and design recovery: a taxonomy. IEEE Software 7(1), 13–17 (1990)
21. Chimia-Opoka, J., Felderer, M., Lenz, C., Lange, C.: Querying UML models using OCL and Prolog: A performance study. In: Proc. of ICSTW, pp. 81–88 (2008)
22. Deutsch, A., Nash, A., Remmel, J.B.: The chase revisisted. In: Proc. of PODS, pp. 149–158 (2008)
23. Donini, F.M., Massacci, F.: EXPTIME tableaux for $\mathcal{ALC}$. Artif. Intell. 124, 87–138 (2000)
24. Dupuy, S., Ledru, Y., Chabre-Peccoud, M.: An Overview of RoZ: A Tool for Integrating UML and Z Specifications. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 417–430. Springer, Heidelberg (2000)
25. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theor. Comput. Sci. 336(1), 89–124 (2005)
26. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. System Sci. 18(2), 194–211 (1979)
27. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)

28. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-based specification environment for validating UML and OCL. Sci. of Computer Progr. 69(1-3), 27–34 (2007)
29. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. Comput. Syst. Sci. 28(1), 167–189 (1984)
30. Kaneiwa, K., Satoh, K.: On the complexities of consistency checking for restricted UML class diagrams. Theor. Comput. Sci. 411(2), 301–323 (2010)
31. Krötzsch, M., Rudolph, S.: Extending decidable existential rules by joining acyclicity and guardedness. In: Proc. of IJCAI, pp. 963–968 (2011)
32. Lenzerini, M., Nobili, P.: On the satisfiability of dependency constraints in entity-relationship schemata. Inf. Syst. 15(4), 453–461 (1990)
33. Lutz, C.: The Complexity of Conjunctive Query Answering in Expressive Description Logics. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 179–193. Springer, Heidelberg (2008)
34. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM Trans. Database Syst. 4(4), 455–469 (1979)
35. Maraee, A., Balaban, M.: Efficient Reasoning about Finite Satisfiability of UML Class Diagrams with Constrained Generalization Sets. In: Akehurst, D.H., Vogel, R., Paige, R.F. (eds.) ECMDA-FA. LNCS, vol. 4530, pp. 17–31. Springer, Heidelberg (2007)
36. Queralt, A., Teniente, E.: Reasoning on UML Class Diagrams with OCL Constraints. In: Embley, D.W., Olivé, A., Ram, S. (eds.) ER 2006. LNCS, vol. 4215, pp. 497–512. Springer, Heidelberg (2006)
37. Queralt, A., Teniente, E.: Verification and validation of UML conceptual schemas with OCL constraints. ACM Trans. Softw. Eng. Meth. 21(2) (2011) (in press)
38. Queralt, A., Teniente, E., Artale, A., Calvanese, D.: OCL-*Lite*: Finite reasoning on UML/OCL conceptual schemas. Data and Know. Eng. (2011) (in press)
39. Richters, M., Gogolla, M.: OCL: Syntax, Semantics, and Tools. In: Clark, A., Warmer, J. (eds.) Object Modeling with the OCL. LNCS, vol. 2263, pp. 447–450. Springer, Heidelberg (2002)
40. Snook, C., Butler, M.: UML-B: Formal modeling and design aided by UML. ACM Trans. Softw. Eng. Meth. 15, 92–122 (2006)
41. Störrle, H.: A PROLOG-based approach to representing and querying software engineering models. In: Proc. of VLL, pp. 71–83 (2007)
42. Vardi, M.Y.: The complexity of relational query languages. In: Proc. of STOC, pp. 137–146 (1982)
43. Vardi, M.Y.: On the complexity of bounded-variable queries. In: Proc. of PODS, pp. 266–276 (1995)