In Proc. of the 2004 Int. Workshop on Description Logics (**DL 2004**)

Towards Implementing Finite Model Reasoning in Description Logics

Marco Cadoli[1], Diego Calvanese[2], Giuseppe De Giacomo[1]

[1] Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
lastname@dis.uniroma1.it

[2] Faculty of Computer Science
Free University of Bolzano/Bozen
Piazza Domenicani 3, I-39100 Bolzano, Italy
calvanese@inf.unibz.it

Abstract

We describe our ongoing work that aims at understanding how one can develop a system that performs finite model reasoning in Description Logics. In particular we report on the preliminary results that we have obtained by encoding reasoning services in DLs as specifications for constraint programming solvers. We have used such an approach to reason with respect to finite models on UML class diagrams.

1 Introduction

Expressive Description Logics (DLs) do not enjoy the finite model property. This means that a knowledge base expressed in such a DL may be satisfiable, though it admits only models with an infinite domain [9, 1]. The loss of the finite model property is due to the interaction between cardinality constraints, the use of direct and inverse roles, and general (possibly cyclic) inclusion assertions in the knowledge base. Hence, for such DLs, techniques have been investigated to address reasoning with respect to finite models only [10, 6, 8, 13]. Notably, the presence of cardinality constraints, together with the requirement of models to be finite, gives rise to combinatorial aspects in the reasoning problems, and such aspects are taken into account by the proposed techniques by resorting to the encoding of satisfiability into solving numerical dependencies.

The recent interest in DLs as a means to formalize and reason on class based formalisms for conceptual modeling, software engineering, and ontologies, has revived the interest in finite model reasoning, since such formalisms are often used to represent structures that are intrinsically finite (e.g., a databases, object repositories, etc.).

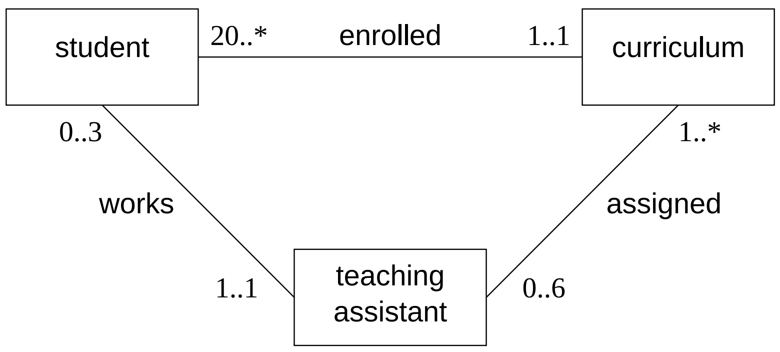

Figure 1: A finitely inconsistent class diagram

In this paper we focus on UML class diagrams.[1] Finite model reasoning in UML class diagrams is of crucial importance for assessing quality of the analysis phase in software development. Finite unsatisfiability of a class diagram means that there are some classes or associations which cannot have a finite number of instances. This concept must be contrasted with unrestricted unsatisfiability, which means that classes or associations cannot have *any*, i.e., both finite and infinite, number of instances. Indeed, for the purpose of software engineering, finite model reasoning in UML class diagrams is often considered more important than unrestricted reasoning.

Example 1 As an example, consider Figure 1, which states reasonable constraints for a university scenario; e.g., there is a binary association *enrolled* between classes *student* and *curriculum*, and the *multiplicities* on the association express that each student is enrolled in exactly one curriculum, while each curriculum must enroll at least 20 students ('*' denotes that there is no constraints on the maximum multiplicity). Such a diagram is satisfiable in the unrestricted sense. Indeed, suppose there is an instance c_1 of *curriculum*; then there must be at least 20 students enrolled in c_1, and since each such student must work with exactly one teaching assistant and each teaching assistant can work with at most 3 students, there must be at least 7 teaching assistants; since at most 6 teaching assistants can be assigned to c_1, there must be a second instance c_2 of *curriculum*; considering that each student can be enrolled in at most one curriculum, by applying the same line of reasoning, we can see that there must be an infinite sequence of instances $c_3, c_4, \ldots$ of *curriculum*. However, this shows also that the diagram in Figure 1 is *not* finitely satisfiable. ■

In this paper we address the implementation of finite model reasoning in DLs, a task that has not been attempted so far. As a matter of fact, state-of-the-art DL reasoning systems, such as FACT or RACER, perform unrestricted reasoning, and do not address finite model reasoning. Interestingly, finite model reasoning on UML class diagrams itself has, to the best of our knowledge, never been implemented in any kind of system.

[1]`http://www.omg.org/uml/`

The main result of this paper is that it is possible to use off-the-shelf tools for constraint modeling and programming for obtaining a finite model reasoner. In particular, exploiting the finite model reasoning technique presented in [10, 8], we propose an encoding as Constraint Satisfaction Problem (CSPs) of knowledge base satisfiability. Moreover, we show also how CSP can be used to actually return a finite model of the knowledge base (a task needed for particular applications, as [4]).

In fact, in this paper, we focus on UML class diagrams without ISA relations between associations, seen as a means of describing DL knowledge bases (see, e.g., [3]). More precisely, such UML class diagrams correspond essentially to *primitive* $\mathcal{ALUNI}$ *knowledge bases*, which consist of (possibly cyclic) inclusion assertions of the form $B \sqsubseteq C$, where B is boolean combination of atomic concepts and C is a concept of the DL $\mathcal{ALUNI}$ [10]. Experimentation so far is in a preliminary stage, but the results we have obtained are quite encouraging.

2 Finite Model Reasoning in DLs

The technique for finite model reasoning in DLs with number restrictions, inverse roles and inclusion assertions was first presented in [10, 6], and is based on translating the knowledge base in a set of linear inequalities. Intuitively, consider a simple knowledge base $\mathcal{K}$ formed by inclusion assertions

$$\begin{array}{rcl} \top & \sqsubseteq & \forall R^{-}.A \sqcap \forall R.B \\ A & \sqsubseteq & (\geq m_1\, R) \sqcap (\leq n_1\, R) \\ B & \sqsubseteq & (\geq m_2\, R^{-}) \sqcap (\leq n_2\, R^{-}) \end{array}$$

for each role R. Such assertions express that R is typed on A for the first component and B for the second, and additionally expresses minimum and maximum cardinality constraints on the participation to R. It is easy to see that such a knowledge base $\mathcal{K}$ is always satisfiable (assuming $m_i \leq n_i$) if we admit infinite models. Hence, only finite model reasoning is of interest. We observe that, if $\mathcal{K}$ is finitely satisfiable, then it admits a finite model in which all atomic concepts are pairwise disjoint. Exploiting this property, we can encode finite satisfiability of $\mathcal{K}$ in a constraint system as follows. We introduce one variable for each role and atomic concept, representing the number of instances of the role (resp., concept) in a possible model of $\mathcal{K}$. Then, for each R we introduce the constraints

$$\begin{array}{rcccl} m_1 \cdot a & \leqslant & r & \leqslant & n_1 \cdot a \\ m_2 \cdot b & \leqslant & r & \leqslant & n_2 \cdot b \\ a \cdot b & \geqslant & r & & \end{array}$$

where a, b, and r are the variables corresponding to A, B, and R, respectively.

It is possible to show that, from a solution of such a constraint system, we can construct a finite model of $\mathcal{K}$ in which the cardinality of the extension of each concept and role is equal to the value assigned to the corresponding variable[2].

[2]In fact, if one is interested just in the existence of a finite model, one could drop the nonlinear constraints of the form $a \cdot b \geqslant r$.

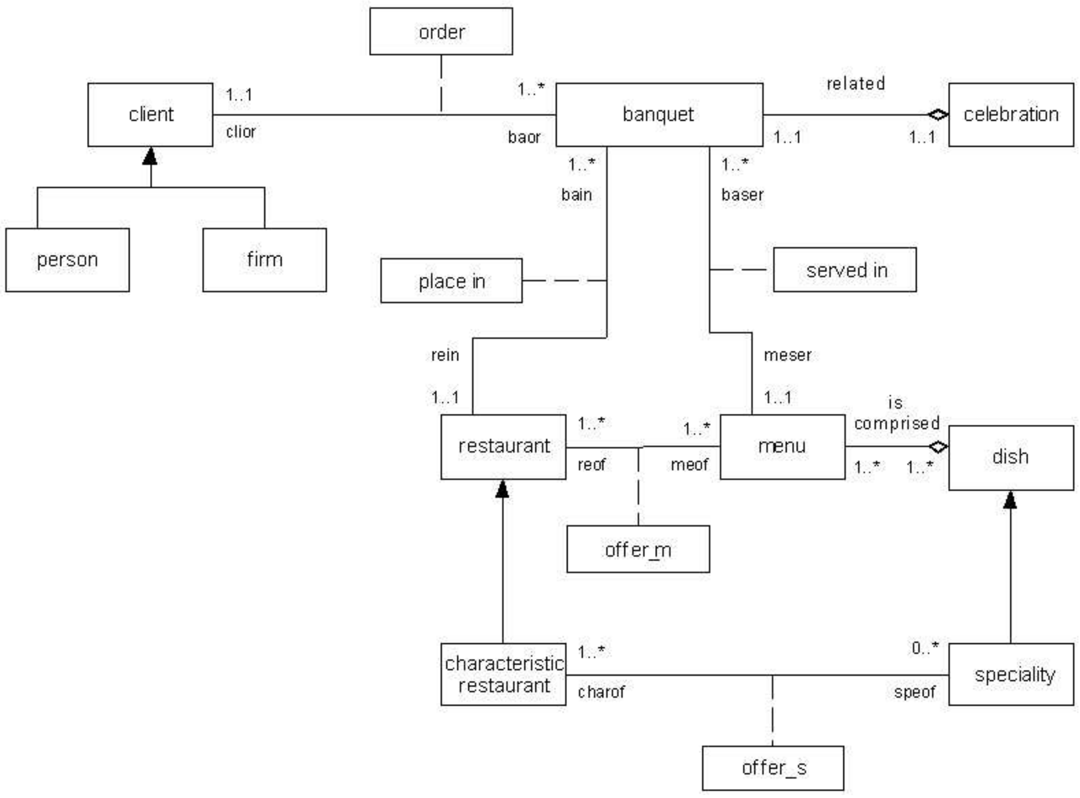

Figure 2: The "restaurant" UML class diagram

The approach above can be extended to deal also with inclusion assertions expressing ISA, disjointness, and covering between concepts. Intuitively, one needs to introduce one variable for each combination of atomic concepts; similarly, for roles one needs to distinguish how, among the possible combinations of atomic concepts, the role is typed in its first and its second component. This leads, in general, to the introduction of an exponential number of variables and constraints [10]. We illustrate this, in the context of UML, in the next section.

3 Finite Model Reasoning via CSP

We address the problem of finite satisfiability of UML class diagrams, and show how it is possible to encode two problems as constraint satisfaction problems (CSPs), namely:

1. deciding whether all classes in the diagram are simultaneously finitely satisfiable, and
2. finding –if possible– a finite model with non-empty classes and associations.

We use the "restaurant" class diagram, shown in Figure 2, as our running example.

First we address the problem of deciding finite satisfiability. As mentioned, we use the technique proposed in [6], which is based on the idea of translating the multiplicity constraints of the UML class diagram into a set of inequalities among integer variables.

The variables and the inequalities of the CSP are modularly described considering in turn each association of the class diagram. Let A be an association between classes C_1 and C_2 such that the following multiplicity constraints are stated:

- there are at least min_1 and at most max_1 links of type A (instances of the association A) for each object of the class C_1;
- there are at least min_2 and at most max_2 links of type A for each object of the class C_2.

Referring to Figure 2, if A stands for *served_in*, C_1 stands for *banquet*, and C_2 stands for *menu*, then min_1 is 1, max_1 is 1, min_2 is 1, and max_2 is ∞.

For the sake of simplicity, we start from the special case in which neither C_1 nor C_2 participates in a ISA hierarchy, e.g., the *related* and the *served_in* associations of Figure 2.

The CSP is defined as follows:

- there are three non-negative variables `c1`, `c2`, and `a`, which stand for the number of objects of the classes C_1 and C_2 and the number of links of A, respectively;
- there are the following constraints (we use the syntax of the constraint programming language OPL [14]):

```
1. min_1 * c1 <= a;
2. max_1 * c1 >= a;
3. min_2 * c2 <= a;
4. max_2 * c2 >= a;
5. a  <= c1 * c2;
6. a  >= 1;
7. c1 >= 1;
8. c2 >= 1;
```

Constraints 1–4 account for the multiplicity of the association; they can be omitted if either `min_`i = 0, or `max_`i = ∞ (symbol '`*`' in the class diagram). Constraint 5 sets an upper bound for the number of links of type A with respect to the number of objects. Constraints 6–8 define the satisfiability problem we are interested in: we want at least one object for each class and at least one link for each association. The latter constraints can be omitted by declaring the variables as strictly positive.

When either C_1 or C_2 are involved in ISA hierarchies, the constraints are more complicated, because the meaning of the multiplicity constraints changes. As an example, the multiplicity `1..*` of the *order* association in Figure 2 states that a *client orders* at least one *banquet*, but the client can be a *person*, a *firm*, both, or neither (assuming the generalization is neither disjoint nor complete). In general, for an ISA hierarchy involving n classes, $O(2^n)$ non-negative variables corresponding to all possible combinations must be considered. For the same reason, we must consider four distinct specializations of the *order* association, i.e., one for each possible combination. Summing up, we have the following non-negative variables:

- `person`, `order_p`, for clients who are persons and not firms;
- `firm`, `order_f`, for clients who are firms and not persons;
- `person_firm`, `order_pf`, for clients who are both firms and persons;
- `client`, `order_c`, for clients who are neither firms nor persons;

plus the positive `banquet` variable.

The constraints (in the OPL syntax) which account for the *order* association are as follows:

```
/*  1 */  client <= order_c;
/*  2 */  firm <= order_f;
/*  3 */  person <= order_p;
/*  4 */  person_firm <= order_pf;
/*  5 */  banquet = order_c + order_f + order_p + order_pf;
/*  6 */  order_c <= client * banquet;
/*  7 */  order_f <= firm * banquet;
/*  8 */  order_p <= person * banquet;
/*  9 */  order_pf <= person_firm * banquet;
/* 10 */  client + firm + person + person_firm >= 1;
/* 11 */  order_c + order_f + order_p + order_pf >= 1;
```

Constraints 1–4 account for the '1' in the `1..*` multiplicity; Constraint 5 translates the `1..1` multiplicity; Constraints 6–9 set an upper bound for the number of links of type *order* with respect to the number of objects; Constraints 10–11 define the satisfiability problem (`banquet` is already strictly positive).

We refer the reader to [8, 6] for formal details of the translation, and in particular for the proof of its correctness. As for the implementation, the "restaurant" example has been encoded in OPL as a CSP with 24 variables and 40 constraints. The solution has been found by the underlying constraint programming solver, i.e., ILOG's SOLVER [12, 11], in less than 0.01 seconds.

4 Constructing a Finite Model

We now turn to the second problem, i.e., finding –if possible– a finite model with non-empty classes and associations. The basic idea is to encode in the constraint modeling language of OPL the semantics of the UML class diagram (see [3, 2]). In particular we use arrays of boolean variables representing the extensions of predicates, where the size of the arrays is determined by the output of the first problem. Since in the first problem we have enforced the multiplicity constraints, and obtained an admissible number of objects for each class (actually for each combination of classes), we know that a finite model of the class diagram exists. We also know the size of the universe of such a finite model, which is equal to the sum, over all combinations of classes, of the number of objects in each combination of classes (recall that each combination of classes is disjoint from all other combinations of classes).

Referring to our "restaurant" example, we have the following declarations describing the size of our universe and two sorts:

```
int size = client + person + firm + person_firm + restaurant + menu +
 characteristic_restaurant + dish + specialty + banquet + celebration;
range Bool 0..1;
range Universe 1..size;
```

The arrays corresponding, e.g., to the *client* and *banquet* classes, and to the *order_c* association are declared as follows:

```
var Bool Client[Universe];
var Bool Banquet[Universe];
var Bool Order_C[Universe,Universe];
```

Now, we have to enforce some constraints to reflect the semantics of the UML class diagram [3, 2], namely that:

1. each object belongs to exactly one class;
2. the number of objects (resp., links) in each class (resp., association) is coherent with the solution of the first problem;
3. the associations are *typed*, e.g., that a link of type *order_c* insists on an object which is a *banquet* and on another object which is a *client*;
4. the multiplicity constraints are satisfied.

Such constraints can be encoded as follows (for brevity, we show only some of the constraints).

```
// AN OBJECT BELONGS TO ONE CLASS
forall(x in Universe)
  Client[x] + Person[x] + Firm[x] + Person_Firm[x] + Restaurant[x] +
  Characteristic_Restaurant[x] + Dish[x] + Specialty[x] + Banquet[x] +
  Celebration[x] + Menu[x] = 1;
// ENFORCING SIZES OF CLASSES AND ASSOCIATIONS
sum(x in Universe) Client[x] = client;
sum(x in Universe) Banquet[x] = banquet;
sum(x in Universe, y in Universe) Order_C[x,y] = order_c;
// TYPES FOR ASSOCIATIONS
forall(x, y in Universe)
  Order_C[x,y] => Client[x] & Banquet[y];
// MULTIPLICITY CONSTRAINTS ARE SATISFIED
forall(x in Universe)
  Client[x] => sum(y in Universe) Order_C[x,y] >= 1;
```

Summing up, the "restaurant" example has been encoded in OPL with about 40 lines of code. After instantiation, this resulted in a CSP with 498 variables and 461 constraints. The solution has been found by ILOG's SOLVER in less than 0.01 seconds, and no backtracking.

5 Notes on Complexity

Few notes about the computational complexity are in order. It is known that solving both problems of deciding finite satisfiability and finite model finding for primitive $\mathcal{ALUNI}$ knowledge bases (and hence for UML class diagrams) are EXPTIME-complete [10, 7, 8]. Our encoding of the first problem in a CSP may result in a number of variables which is exponential in the size of the diagram. Anyway, since the exponentiality depends on the maximum number of classes involved in the same ISA hierarchy, the actual size for real UML diagrams will not be very large.

As for the second problem, our encoding is polynomial in the size of the class diagram. Note that this does not contradict the EXPTIME lower bound, due to the *program complexity* of modeling languages such as OPL. Indeed, in [5] it is shown that the program complexity of boolean linear programming is NEXPTIME-hard.

6 Conclusions

In this paper we have reported on our ongoing investigation on implementing finite model reasoning in DLs. We have shown how current state-of-the-art constraint solvers can be used to perform such a kind of reasoning. The performance of such systems in the experiments done so far is quite good, though these results still need to be confirmed on larger cases, such as the CIM[3] diagrams. This is ongoing work. We are also building a prototype system for finite model reasoning that uses OPL as reasoning engine.

Our implementation can so far not deal with UML class diagrams containing associations between roles. The translation of such diagrams in a DL knowledge base requires to introduce inclusion assertions between roles, or, alternatively, to reify roles and thus introduce qualified number restrictions to encode multiplicities [2]. In [8, 6], an extension of the technique for finite model reasoning illustrated here is proposed, that can deal also with qualified number restrictions. However, such a method requires to introduce a number of variables and constraints that is double exponential in the size of the knowledge base. In [13] a more involved technique is presented, for which the size of the constraint system stays simply exponential. However, numbers appearing in number restrictions cannot be dealt with directly in the constraints and need to be encoded using counters. Thus, the possibility of actually implementing one or the other of these methods by making use of constraint solvers requires further investigation.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2003.

[3]`http://www.dmtf.org/standards/cim/`

[2] D. Berardi, A. Calì, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. Technical Report 11-03, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 2003.

[3] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams is EXPTIME-hard. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*, pages 28–37. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-81/`, 2003.

[4] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. *e*-Service composition by description logics based reasoning. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*, pages 75–84. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-81/`, 2003.

[5] M. Cadoli. The expressive power of binary linear programming. In *Proc. of the 7th Int. Conf. on Principles and Practice of Constraint Programming (CP 2001)*, volume 2239 of *Lecture Notes in Computer Science*, 2001.

[6] D. Calvanese. Finite model reasoning in description logics. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 292–303, 1996.

[7] D. Calvanese. Reasoning with inclusion axioms in description logics: Algorithms and complexity. In *Proc. of the 12th Eur. Conf. on Artificial Intelligence (ECAI'96)*, pages 303–307. John Wiley & Sons, 1996.

[8] D. Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms.* PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1996. Available at `http://www.dis.uniroma1.it/pub/calvanes/thesis.ps.gz`.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, and D. Nardi. Reasoning in expressive description logics. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 23, pages 1581–1634. Elsevier Science Publishers (North-Holland), Amsterdam, 2001.

[10] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class based representation formalisms. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 109–120, 1994.

[11] ILOG Solver system version 5.1 user's manual, 2001.

[12] ILOG OPL Studio system version 3.6.1 user's manual, 2002.

[13] C. Lutz, U. Sattler, and L. Tendera. The complexity of finite model reasoning in description logics. In *Proc. of the 19th Int. Conf. on Automated Deduction (CADE 2003)*, pages 60–74, 2003.

[14] P. Van Hentenryck. *The OPL Optimization Programming Language.* The MIT Press, 1999.