A UML-Based Method for Deciding Finite Satisfiability in Description Logics*

Azzam Maraee and Mira Balaban

Computer Science Department
Ben-Gurion University of the Negev, Beer-Sheva 84105, ISRAEL
mari@bgu.ac.il, mira@cs.bgu.ac.il

Abstract. *Finite satisfiability* in Description Logics and in UML class diagrams is the problem of deciding whether a concept (a class) has a finite, non-empty extension in some model. The problem is known to be hard. Standard DL reasoners do not reason about finite satisfiability. In this article we introduce class diagram translations for major operators in description logics, and extend a previous UML finite satisfiability decision algorithm to handle these translations.
The contribution of this article is in presenting an efficient method for deciding finite satisfiability in atomic, primitive knowledge bases of popular description logics, using a translation to UML class diagrams. The method applies to class hierarchies that do not include cycles with *disjoint* or *complete* constraints. The scope can be determined in a preprocessing step. The suggested method is valuable since standard DL reasoners do not reason about finite satisfiability,

Keywords:
Description logics, multiplicity constraint operators, UML class diagram, finite satisfiability, multiplicity constraints, class hierarchy constraints, generalization set constraints, association class constraints, class hierarchy structure, linear programming reduction.

1 Introduction

Finite satisfiability in Description Logics (DLs) is the problem of deciding whether a knowledge base has a finite, non-empty model. The problem is known to be hard: EXPTIME-complete for the ALC DL [10] and EXPTIME for the $ALCQI$ DL [15]. Standard DL reasoners do not reason about finite satisfiability.

Class based representations, like UML class diagrams and Entity-Relationship (DL) diagrams, have a similar *finite satisfiability* problem: Whether a diagram has a finite and non-empty instance[1] Figure 1-a, presents a small ontology in

* Supported by the Lynn and William Frankel center for Computer Sciences.

[1] The problem requires that for every class there is an instance in which it has a finite, non-empty extension. But it can be shown that for UML class diagrams, this implies having a legal instance in which all class extensions are finite and non-empty [11, 13, 12]. Such instances are called *finite, non-empty instances.*

the Molecular Biology domain. It includes a multiplicity and hierarchy constraint cycle that involves the classes *Chemical, Reaction, CatalyzedReaction, Enzyme* and *Protein*. Instances of *Chemical* must be related to *Reaction* instances, which are also instances of *CatalyzedReaction*, whose instances must be related, each, to two *Enzyme* instances, which are also instances of *Protein* and of *Chemical*. Careful analysis reveals that in every non-empty finite instance of this diagram the number C of *Chemical* instances must satisfy the inequality $C < C$. Indeed, the class diagram is *consistent*, i.e., has a non-empty instance[2], but in every instance, *Chemical* denotes either an empty or an infinite set, implying that the class diagram is not *finitely satisfiable*.

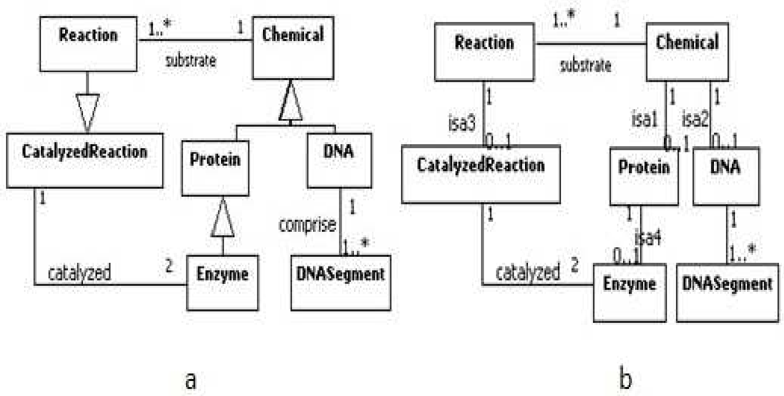

Fig. 1. A Class Diagram with a Finite Satisfiability Problem

Berardi et al. [2], provide finite model preserving reductions of the ALC description logic into UML class diagrams, and of a restricted version (minor restrictions) of the latter into the description logic $ALCQI$. Therefore, finite satisfiability of UML class diagrams is also EXPTIME-complete.

Nevertheless, there are several methods for deciding finite satisfiability in fractions of UML class diagrams. There are two main approaches for reasoning about finite satisfiability of class diagrams: The *linear inequalities approach* and the *graph based approach* [11, 14, 8, 9]. The first approach reduces the finite satisfiability problem to the problem of finding a solution to a system of linear inequalities. The second approach detects infinity causing cycles in the diagram, and possibly suggests repair transformations. All methods apply only to fragments of UML class diagrams. Deciding infinity in unrestricted class diagrams is still an open issue.

[2] The problem requires that for every class there is an instance in which it has a non-empty extension. But it can be shown that for UML class diagrams this implies having a legal instance in which all class extensions are non-empty. Such instances are called *non-empty instances*.

The fundamental work in the linear inequalities approach is that of [11, 6] (also [1, 3, 14]). It applies to *Entity-Relationship* (*ER*) diagrams with binary *multiplicity constraints*. The method transforms the multiplicity constraints into a system of linear inequalities whose size is polynomial in the size of the diagram. Calvanese and Lenzerini, in [5], extend this method to apply to diagrams with class hierarchy constraints. However, the size of the resulting system of inequalities is exponential in the size of the class diagram.

Maraee and Balaban also extend the [11] method to handle class diagrams with class hierarchy constraints and generalization set constraints [13, 12]. This method, termed the ***FiniteSat*** algorithm, has a polynomial complexity, but its scope is limited to class hierarchies without undirected cycles with *disjoint* or *complete* constraint. This scope can be determined by a preprocessing procedure.

In this article we extend the ALC to class diagrams translation of [2] to include multiplicity constraint operators in description logics, and extend the ***FiniteSat*** algorithm [13] to handle association class constraints. The extended version handles all of the constraints used in the description logic translations. Therefore, the extended ***FiniteSat*** algorithm can be used to decide finite satisfiability in description logic knowledge bases, provided that their class hierarchies fall in the scope of ***FiniteSat***.

The contribution of this article is in presenting an efficient method for deciding finite satisfiability in atomic, primitive knowledge bases of popular description logics, using a translation to UML class diagrams. The method applies to class hierarchies that do not include undirected cycles with *disjoint* or *complete* constraints. The scope can be determined in a preprocessing step.. The suggested method is valuable since standard DL reasoners do not reason about finite satisfiability.

Section 2 describes a finite model preserving reduction of a DL knowledge base into a UML class diagram. Section 3 describes an extension of our previous work, for deciding finite satisfiability in UML class diagrams with constrained generalization sets and association classes. Section 4 points to future extensions.

2 DLs-to-UML

Berardi et al., in [2], showed that deciding consistency of UML class diagrams is EXPTIME-complete. The proof is obtained by providing reductions to/from hard *Description Logics*. They show that the description logic ALC can be encoded by class diagrams, and class diagrams can be encoded in the description logic DLR_{ifd}. The reductions preserve consistency, finite satisfiability and logical implication. For the purpose of this work, we are interested in reductions from description logics into UML class diagrams.

The ALC into UML reduction of [2] is based on a translation of an *atomic primitive* ALC formula into class diagrams ("atomic" means that the subsumed concept in a formula is atomic, and "primitive" means that there is no operator nesting). The formulae translated in [2] are: $A \sqsubseteq \neg B$, $A \sqsubseteq B_1 \sqcup B_2$, $A \sqsubseteq \forall R.B$, $A \sqsubseteq \exists R.B$. The reduction cannot be easily extended to non-atomic or non-primitive formulae since it is not compositional: The translation is applied to formulae, which cannot be composed (as opposed to concepts or roles).

The ALC operators do not include multiplicity constraints. Since finite satisfiability problems are caused by cycles of multiplicity constraints, non-finite satisfiability in ALC is caused by inconsistency, i.e., lack of non-empty models, and not by lack of finite models. Therefore, we develop consistency and finite satisfiability preserving translations of atomic, primitive DL formulae with multiplicity constraints, and show how our method can be applied for deciding finite satisfiability in description logics that include these operators.

Translation of atomic, primitive DL formulae involving multiplicity constraints and role inverse operators: The translations are inspired by the translations in [2].

1. $A \sqsubseteq B_1 \sqcap B_2$: The translation in given in Figure 2-a.
2. $A \sqsubseteq \leq nR$ ($A \sqsubseteq \geq nR$, $A \sqsubseteq = nR$): These formulae state that the concept A is subsumed by the concept of all entities that are related via the role R to at most (at least, exactly) n entities. The translation in Figure 2-b splits the R association class, that stands for the R role, into disjoint sub-classes, $R\_\bar{A}$ and R_A, that cover R and identify all R pairs that start in $\bar{A}$, and in A, respectively. Every A entity has at most n R links since the pairs in $R\_\bar{A}$ do not start in A entities. In order to account for the $\geq$ and the $=$ operators, the multiplicity constraint on the R_A association should be changed to $n..*$ and n, respectively.

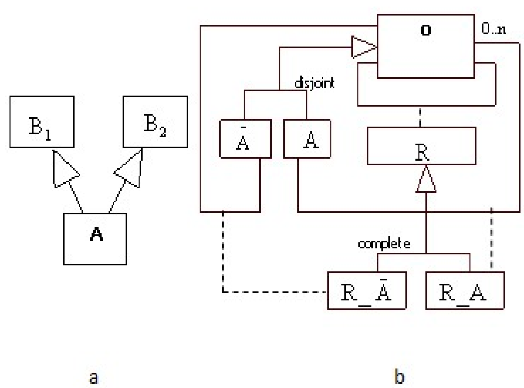

Fig. 2. The Transformation of $A \sqsubseteq B_1 \sqcap B_2$ an d$A \sqsubseteq \leq nR$ to UML

3. $A \sqsubseteq \leq nR.B$ ($A \sqsubseteq \geq nR.B$, $A \sqsubseteq = nR.B$): These formulae state that the concept A is subsumed by the concept of all entities that are related via the role R to at most (at least, exactly) n entities in B. The translation in Figure 3 splits the association class R_A in a similar way as above. R_A is split into disjoint sub-classes, $R\_A\bar{B}$ and R_AB, that cover R_A and identify all R_A pairs that end in $\bar{B}$, and in B, respectively. Every A entity has at

most n R links in B since the pairs in $R\_A\bar{B}$ do not and in B entities. In order to account for the $\geq$ and the $=$ operators, the multiplicity constraint on the R_AB association should be changed to $n..*$ and n, respectively.

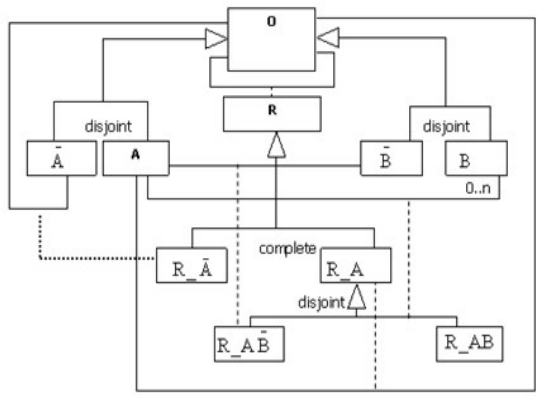

Fig. 3. The Transformation of $A \sqsubseteq \leq nR.B$ to UML

4. R^-: In order to account for the R^- operator, there is a need to translate the above multiplicity constraint formulae with respect to R^-. The translations are given by the above UML class diagrams, with the single modification that the multiplicity constraints are moved to the A class association end.

Claim. An atomic concept A is finitely satisfiable in an atomic primitive DL knowledge base, if and only if class A is finitely satisfiable in the UML class diagram, constructed by the above translation.

3 Efficient Decision of Finite Satisfiability in UML Class Diagrams

The Lenzerini and Nobili method [11] applies to *Entity-Relationship* (*ER*) diagrams with *Entity Types* (*Classes*), *Binary Relationships*[3]. (*Associations*), and *multiplicity Constraints.* The method transforms the multiplicity constraints into a system of linear inequalities whose size is polynomial in the size of the diagram:

1. For every entity or association type T, insert a variable t and the inequality: $t > 0$.
2. For multiplicity constraints $r(rn_1 : C_1[min_1, max_1], rn_2 : C_2[min_2, max_2])$, (imposed on an association r between classes C_1 and C_2, with roles named rn_1 and rn_2, respectively), insert the inequalities:

[3] They allow also n-ary relationships, but with non-standard (membership) semantics for cardinality constraints.

- For $min_2 > 0: \ \ r \geq min_2 \cdot c_1$ and for $max_2 \neq *: \ \ r \leq max_2 \cdot c_1$.
- For $min_1 > 0: \ \ r \geq min_1 \cdot c_2$ and for $max_1 \neq *: \ \ r \leq max_1 \cdot c_2$.

Calvanese and Lenzerini, in [5], extend the inequalities based method of [11] to apply to diagrams with class hierarchy constraints. The expansion introduces a variable for every possible class intersection among subclasses of a superclass, and splits relationships accordingly. Therefore, the size of the resulting system of inequalities is exponential in the size of the class diagram. The simplification of [4] reduces the overall number of new class and association variables, but the worst case is still exponential.

In this section we describe the ***FiniteSat*** efficient algorithm for deciding finite satisfiability in UML class diagrams. The scope of the algorithm is defined by the structure of the class hierarchy in a knowledge base, rather than by a fragment of the language. First, we present an improved version of the [13] algorithm, which applies to class diagrams with *multiplicity constraints on binary associations, class hierarchy constraints*, and *Generalization Set* (GS) *constraints*. Then, we extend the algorithm to handle also *Association Class* (AC) *constraints*. Together with this extension, this algorithm can be used for deciding finite satisfiability in atomic, primitive DL knowledge bases that include formula with the operators that are translated in the previous section.

3.1 The *FiniteSat* Algorithm

The ***FiniteSat*** algorithm extends the algorithm of Lenzerni and Nobili [11] to handle also class hierarchy constraints and GS constraints. The version presented below improves the [13] version by having a single stage (omitting the intermediate stage of [13]), and producing a simpler inequality system.

Algorithm 1 ***The FinitSat Algorithm***
Input*: A class diagram CD with binary multiplicity constraints, class hierarchy constraints, and GS constraints.*
Output*: A linear inequality system Ψ^{CD}*
Method*:*

1. *For every class, association, or multiplicity constraint, create variables and inequalities according to the Lenzerini and Nobili method.*
2. *For every class hierarchy $B \prec A$ constraint, B being the subclass with variable b, and A being the super class with variable a, extend the inequality system with the inequalities $a \geq b$.*
3. *For every GS constraint $GS(C, C_1, ...C_n;\ Const)$, C being the super-class, $C_i s$ being the subclasses, and $Const$ being the GS constraint, extend the inequality system, as follows:*
   - *$Const$ = disjoint: $C \geq \sum_{j=1}^{n} C_j$*
   - *$Const$ = complete: $C \leq \sum_{j=1}^{n} C_j$*
   - *$Const$ = incomplete: $\forall j \in [1, n].C > C_j$*
   - *$Const$ = overlapping: Without inequality*
   - *disjoint, incomplete: $C > \sum_{j=1}^{n} C_j$.*

- *disjoint, complete:* $C = \sum_{j=1}^{n} C_j$.
- *overlapping, complete:* $C < \sum_{j=1}^{n} C_j$.
- *overlapping, incomplete:* $\forall j \in [1, n].C > C_j$.

Proving the correctness of the ***FiniteSat*** algorithm requires analysis of the structure of class hierarchies. For that purpose, we consider the graph of class hierarchy constraints alone, in which nodes represent classes and directed edges represent ISA constraints, directed from superclasses to their subclasses (association lines are removed). We consider two versions of such graphs: Directed and undirected. Three class hierarchy structures are analyzed:

1. *Tree class hierarchy*: The directed graph of the class hierarchy forms a tree, as in Figure 1.
2. *Acyclic class hierarchies*: The undirected graph of the class hierarchy is acyclic. In Figure 4-a, the directed class hierarchy is not a tree, as F is a sub class of both C and D, but the undirected class hierarchy graph is acyclic (a tree).
3. *Cyclic class hierarchies*: The undirected graph of the class hierarchy is cyclic. Multiple inheritance is unrestricted, as the undirected induced graph can be cyclic. In Figure 4-b, class F has two ISA paths to its superclass A. The ISA path A, B, F, C, A forms an undirected ISA cycle.

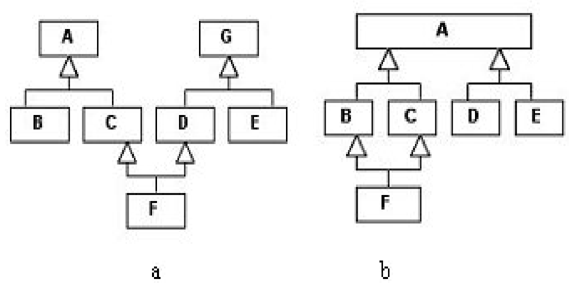

Fig. 4. Unconstrained Hierarchy Structures

The correctness of Algorithm ***FiniteSat*** is proved via a reduction of the finite satisfiability of a class diagram CD to the finite satisfiability of a class diagram CD', that does not include class hierarchies, and therefore, the [11] method applies to it. CD' is created as follows: Initialize CD' by CD. Replace all class hierarchy constraints with new regular binary associations (termed henceforth ISA associations) between the superclass to the subclasses. The multiplicity constraints on these associations are 1..1 participation constraint for the subclass (written on the super class end in the diagram) and 0..1 participation constraint for the super class. Figure 1-b shows the reduced class diagram of Figure 1-a.

Lemma 1. *Finite satisfiability of CD is reducible into the finite satisfiability of CD'.*

Proof. (Sketched) The reduction is defined by bi-directional translations between non-empty finite legal instances I and I' of CD and CD', respectively. The translations rely on a mapping T (and its inverse T^{-1}) from I' to I, which collapses a structure of ISA-linked objects in I' into a single object in I. The intuition is that CD' splits a single instance object of CD into its components in its ancestor classes.

A crucial property of the T translation is that ISA-linked objects in I' should not include two objects from the same class. This property, termed the *Single Class property*, ensures that the T mapping maps an instance I' of CD' to an instance I of CD. The main problem is showing that the mapping preserves multiplicity constraints (otherwise, while collapsed into a single object in I, the links of two objects are combined into links of a single one). Full proof appears in [12].

The reduction is proved by considering the three forms of class hierarchy graphs. For trees and for acyclic hierarchies, the single class property holds for every instance. For cyclic class hierarchies, it is shown that if a diagram is finitely satisfiable, then it has an instance that satisfies the single class property.

Claim 1 (*FiniteSat* correctness – without GS constraints) *A class diagram with binary multiplicity constraints and class hierarchy constraints is finitely satisfiable if and only if the inequality system constructed by Algorithm **FiniteSat** is solvable.*

Proof. (Sketched) Given a class diagram CD, construct a class diagram CD' as above, to which the inequalities method of [11] is applied. Based on Lemma 1, CD is finitely satisfiable if and only if the inequality system of [11] for CD' is solvable. It is not hard to show that this inequality system is equivalent to the inequality system constructed by ***FiniteSat***.

Claim 2 (*FiniteSat* correctness – GS constraints, acyclic hierarchy) *A class diagram with binary multiplicity constraints, class hierarchy constraints, and GS constraints, with a tree or acyclic class hierarchy structure is finitely satisfiable if and only if the inequality system constructed by Algorithm **FiniteSat** is solvable.*

Proof. (sketched) The CD' diagram must be constrained by the GS constraints – otherwise, there is no reduction of the finite satisfiability problem. For each GS constraint, it is shown that every solution for the inequality system can induce an instance of the constrained CD' that satisfies the single class property.

Claim 3 (*FiniteSat* correctness – GS constraints, cyclic hierarchy) *A class diagram with binary multiplicity constraints, class hierarchy constraints, and GS constraints, in which class hierarchy cycles do not include disjoint or complete constraints, is finitely satisfiable if and only if the inequality system constructed by Algorithm **FiniteSat** is solvable.*

The scope of the ***FiniteSat*** algorithm is defined in the following claim:

Claim 4 (Partial correctness – GS constraints, cyclic hierarchy) *A class diagram with binary multiplicity constraints, class hierarchy constraints, and GS constraints, in which class hierarchy cycles include disjoint or complete constraints, is not-finitely satisfiable if the inequality system constructed by Algorithm* ***FiniteSat*** *is not solvable.*

Proof. In cyclic class hierarchies, the *disjoint* or *complete* *GS*-constraints might have an implicit global effect on other generalization sets in a cycle. Therefore, if the inequality system does not have a solution, the corresponding diagram does not have a legal finite non-empty instance, but a solution for the inequalities might miss the implicit constraints.

Table 1 summarizes our results of the above claims.

Graph Structure	With/ Without GS constraints	***FiniteSat*** correctness
Acyclic	Without	**correct**
	with	**correct**
Cyclic	Without	**correct**
	No *disjoint* or *complete* in cycles	**correct**
	disjoint or *complete* in cycles	**partially correct**

Table 1. The Scope of The ***FiniteSat*** Algorithm

Claim 5 (Complexity of the *FiniteSat* algorithm) *The construction of the inequalities by* ***FiniteSat****, and their number is $O(n)$, where n is the number of constraints in the class diagram.*

Proof. Every constraint contributes a constant number of inequalities.

3.2 Extension of *FiniteSat* to Handle Association Class Constraints

Association classes are classes whose instances are identified by tuples of the associated associations. That is, in every instance of the class diagram, there is a $1:1$ and onto mapping between the extensions of an association r and its associated association class AC^r. The semantics of UML dictates that a class hierarchy constraint between association classes $AC^r \prec AC^q$ requires that the sub-association r satisfies all of the constraints imposed on the super-association q (note that this does not necessarily enforces a subset constraint between r and q). In particular, this applies to association multiplicity constraints[4].

[4] The logic based semantics that Berardi et al. [2] provide for UML class diagrams seems to overlook the implication of association class hierarchy on their associated

Extending ***FiniteSat*** to account for association class constraints involves the following addition of step (4), which accounts for the identification of the association class with its association, and for the inheritance of the multiplicity constraints:
***FiniteSat* Algorithm- Extension to Association Class**:

4 For every association class AC^r:

1. Extend the inequality system with the equality: $AC^r = r$, assuming that AC^r and r are the variables of AC^r and of r, respectively.
2. For every association class AC^q, such that $AC^r \prec^* AC^q$, let r inherit all the multiplicity constraints of q. That is, if q has the multiplicity constraint $q(qn_1 : C_1[min_1, max_1],\ qn_2 : C_2[min_2, max_2])$, and r has the constraint $r(rn_1 : C'_1[min'_1, max'_1],\ rn_2 : C'_2[min'_2, max'_2])$, where the roles $qn_1,\ qn_2$ correspond to the roles $rn_1,\ rn_2$, respectively, then, add the new multiplicity constraint on r, $r(qn_1 : C'_1[min_1, max_1],\ qn_2 : C'_2[min_2, max_2])$, and apply the Lenzerini and Nobili construction to the new constraint.

Claim 6 (Complexity of the extended*FiniteSat* algorithm) *The construction of the inequalities by the extended version of* ***FiniteSat****, and their number is $O(n^2)$, where n is the number of constraints in the class diagram.*

Proof. Each association class hieratchy constraints requires going over the whole hierarchy.

Theorem 1 **(*FiniteSat* correctness).** *A class diagram CD with binary multiplicity constraints, class hierarchy constraints, GS constraints, and association class constraints:*

1. *If the class hierarchy structure does not include cycles with a disjoint or complete, then CD is finitely satisfiable if and only if Ψ^{CD} is solvable.*
2. *If the class hierarchy structure includes cycles with a disjoint or a complete constraints, then CD is not-finitely satisfiable if Ψ^{CD} is not solvable.*

4 Future Work

Using UML class diagrams for obtaining DL services is quite new. The more conventional direction is the other way around. The translations, in both directions, emphasize the representational merits of both languages.

An implementation for the ***FinitSat*** algorithm is on the way, and would enable us to test the usability of the suggested approach. Still, in view of the

associations. Their semantics requires only the $1:1$ mapping between an association class extension to its associated association extension. Consequently, their translations of UML class diagrams to DLR_{ifd} and to $ALCQI$ cannot infer inheritance of multiplicity constraints, between associations whose association classes are constrained by class hierarchy cumbersome translations, it makes sense to try and develop a direct DL version of the ***FinitSat*** algorithm.

Another direction is to strengthen the DL to UML translation, to apply to concepts and roles, rather than to formulae. Such a translation can be compositional and apply to non primitive knowledge bases.

References

[1] Balaban, M., Shoval, P.: MEER-An EER Model Enhanced with Structure Methods. Information Systems, Volume 27, Issue 4 (2002).

[2] Berardi, D., Calvanese, D., Giacomo, De.: Reasoning on UML class diagrams. Artificial Intelligence (2005).

[3] Boufares, F., Bennaceur, H.: Consistency Problems in ER-schemas for Database Systems. Information Sciences, Issue 4 (2004).

[4] Cadoli, M., Calvanese, D, De Giacomo, G, Mancini, T.: Finite Satisfiability of UML Class Diagrams by Constraint Programming. In Proc. of the CP 2004 Workshop on CSP Techniques with Immediate Application, (2004).

[5] Calvanese, D., Lenzerini, M.: On the Interaction between ISA and Cardinality Constraints. Proc. of the 10th IEEE Int. Conf. on Data Engineering (1994).

[6] Thalheim, B.:Fundamentals of cardinality constraints. Proc. of The 11th International Conference on the Entity-Relationship Approach (1992).

[7] Diego Calvanese.: Finite model reasoning in description logics. In Luigia C. Aiello, John Doyle, and Stuart C. Shapiro, editors, Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-96). Morgan Kaufmann, Los Altos, (1996).

[8] Hartman, S.: Graph Theoretic Methods to Construct Entity-Relationship Databases. LNCS, Vol. 1017, (1995).

[9] Hartman, S.: Coping with Inconsistent Constraint Specifications. LNCS, Vol. 2224, (2001).

[10] Schild, A. : A correspondence Theory for Terminological logics:Preliminary Report. Technical report KIT-BACK, FR 5-12, (1991).

[11] Lenzerini, M. Nobili, P.: on the Satisfiability of Dependency Constraints in Entity-Relationship Schemata. Information Systems, Vol. 15, 4, (1990).

[12] Maraee, A.: Efficient Methods for Solving Finite Satisfiability Problems in UML class Diagrams. Master' thesis, Ben-Gurion University of the Negev (2007).

[13] Maraee, A., Balaban, M.: Efficient Reasoning About Finite Satisfiability of UML Class Diagrams with Constrained Generalization Sets. The 3rd European Conference on Model-Driven Architecture, June 2007, Haifa Israel.

[14] Thalheim, B.: Entity Relationship Modeling, Foundation of Database Technology. Springer-Verlag, (2000).

[15] Lutz, C., Sattler, U., and Tendera, L. 2005. The complexity of finite model reasoning in description logics. Inf. Comput. 199, 1-2 (May. 2005).