

Efficient Recognition and Detection of Finite Satisfiability Problems in UML Class Diagrams: Handling Constrained Generalization Sets, Qualifiers and Association Class Constraints

Azzam Maraee, Victor Makarenkov, Mira Balaban * **

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

Abstract. Models lie at the heart of the emerging Model-driven Engineering approach. In order to guarantee precise, consistent and correct models, there is an urgent need for efficient methods for verifying model correctness. Class diagrams are the most important UML model. *Finite satisfiability* of class diagrams characterizes the ability to finitely instantiate the classes of a class diagram, without violating any constraint. This paper extends our previous work on efficient recognition of finite satisfiability problems in UML class diagrams with constrained generalization sets. Previous results are strengthened to handle qualifier and association class constraints. We also present a method for detection of the constraints that cause finite satisfiability.

Keywords:

UML class diagram, finite satisfiability, recognition and detection, multiplicity constraints, class hierarchy constraints, generalization set constraints, qualifier constraints, association class constraints, class hierarchy structure, linear programming reduction, critical cycles.

1 Introduction

Models lie at the heart of the emerging Model-driven Engineering approach, in which software is developed by repeated transformations of models. In this paradigm, models are no longer restricted design artifacts, but play a central role in the process of software development. Therefore, it is essential to have precise, consistent and correct models. Yet, current case tools enable the construction of erroneous models. There is an urgent need for efficient methods for verifying model correctness.

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

** This work was supported in part by the Paul Ivanir Center for Robotics and Production Management at Ben-Gurion University of the Negev. Contact: POB 653, Beer Sheva 84105, ISRAEL, Phone: +972-8-6472222, FAX: +972-8-6477527

The Unified Modeling Language (UML) is nowadays the widely accepted standard for modeling systems. It consists of a variety of visual modeling diagrams, each describing a different view of object-oriented software. *Class Diagrams* are probably the most important and best understood among all UML models. A class diagram provides a static description of system components. It describes system structure in terms of classes, associations, and constraints.

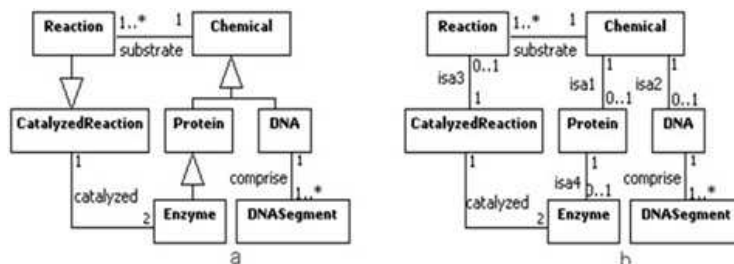


Fig. 1. A Class Diagram with a Finite Satisfiability Problem

Class diagrams are models written by people, and therefore, they cannot guarantee correctness and consistency. Problems usually arise in the presence of constraints, which may turn the model inconsistent, or may not be satisfied. A class diagram is *finitely satisfiable* if its classes can be finitely instantiated, without contradicting the constraints. Deciding finite satisfiability of UML class diagrams is known to be an EXPTIME-complete problem [5].

Figure 1-a, presents a small ontology in the Molecular Biology domain (deliberately spoiled to create a finite satisfiability problem). 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 E of *Enzyme* instances and the number R of *Reaction* instances, must satisfy the relationships $E \geq 2R$ and $E \leq R$. Indeed, the class diagram is *consistent*, i.e., has a non-empty instance¹, but in every instance, *Reaction* and *Enzyme* denote either empty or infinite sets, implying that the class diagram is not *finitely satisfiable*.

This paper extends our previous work [1] on efficient recognition of finite satisfiability problems in UML class diagrams with constrained generalization sets. We strengthen our previous results by extending the scope of class diagrams

¹ 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*.

to which our methods apply to include qualifier and association class constraints. Furthermore, we add a method for detecting the constraints that cause a finite satisfiability problem.

Section 2 provides the relevant terminology; Section 3 describes our earlier work; Section 4 presents the extension to handle qualifier constraints, and Section 5 describes the extension to handle association class constraints. Finally, Section 6 describes the cause detection method. Section 7 concludes the paper.

2 Background

The subset of UML2.0 class diagrams considered in this paper includes *classes* with attributes, *associations* and five kinds of constraints:

1. *Multiplicity (cardinality)* constraints on binary associations, denoted symbolically as $a(rn_1 : C_1[min_1, max_1], rn_2 : C_2[min_2, max_2])$, where a is an association between classes C_1, C_2 , with roles rn_1, rn_2 , and multiplicity constraints $[min_i, max_i]$, $i = 1, 2$ (written on the rn_i association end).
2. *Class hierarchy* constraints, denoted symbolically $C_2 \prec C_1$, where C_1 is the super class and C_2 is the subclass (also called *direct descendant*).
3. *Generalization set (GS)* constraints, denoted symbolically $GS(C, C_1, \dots, C_n; constraint)$, where *constraint* is one of *complete, incomplete, disjoint, overlapping*, C is the super class and the C_i -s are the subclasses ($C \neq C_i$, for $i = 1, \dots, n$). An unconstrained *GS* specifies only class hierarchy constraints. *GS*s are visualized as in Figure 1. *GS* constraints can be combined as follows: $\{complete, disjoint\}$, $\{incomplete, disjoint\}$, $\{complete, overlapping\}$, $\{incomplete, overlapping\}$.
4. *Qualifier* constraints, that strengthen multiplicity constraints.
5. *Association class* constraints, that reify associations, and are denoted AC^a for an association a .

A class diagram CD' is a *sub-diagram* of a class diagram CD , denoted $CD' \leq CD$, if its classes, associations and constraints belong to CD .

The standard set theoretic semantics of class diagrams associates a class diagram with *instances* in which class extensions are sets of objects and association extensions are relationships among class extensions. Class hierarchy constraints denote subset relations. *GS* constraints have the following meaning (class symbols are identified with their extensions):

1. *complete*: $C = \bigcup_{i=1}^n C_i$
2. *incomplete*: $\bigcup_{i=1}^n C_i \subset C$
3. *disjoint*: $C_i \cap C_j = \emptyset, \forall i, j$.
4. *overlapping*: For some i, j , $C_i \cap C_j \neq \emptyset$.

A *legal instance* of a class diagram is an instance that satisfies all constraints. Correctness of a class diagram involves *consistency* and *finite-satisfiability* [2, 3, 4, 5, 1]. A *class is consistent* if it has a non-empty extension in some legal instance; a *class diagram is consistent* if all of its classes are consistent; a *class is finitely satisfiable* if it has a non-empty extension in some legal finite instance; a *class diagram is finitely satisfiable* if all of its classes are finitely satisfiable². It can be shown that a consistent class diagram has a legal instance in which all class extensions are non-empty, and a finitely satisfiable diagram has a legal instance in which all class extensions are non-empty and finite [6]. Class diagrams CD , CD' are *equivalent*, denoted $CD \equiv CD'$, if they have the same legal instances.

Complexity: Berardi et al., in [5], showed that deciding consistency of UML class diagrams is EXPTIME-complete. Artale et al. [7] refine these results, by considering fragments of class diagrams. They show that for ER diagrams that include, besides cardinality, class hierarchy and *disjoint* constraints, deciding consistency is in NLogSpace. Addition of *complete* constraints raises the complexity to NP, and addition of association hierarchy has already the EXPTIME-complete complexity. Recently, it was shown [8, 9] that finite satisfiability of the description logic *ALCQI* is EXPTIME-complete, which implies that finite satisfiability of class diagrams (under some minor restrictions) is also EXPTIME-complete.

Reasoning about Finite Satisfiability 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*. 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 finite satisfiability in unrestricted class diagrams is still an open issue. Below, we shortly summarize results in both approaches, on which our research is based.

The fundamental work in the linear inequalities approach is that of [3, 10]. It 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:
 - For $min_2 > 0$: $r \geq min_2 \cdot c_1$ and for $max_2 \neq *$: $r \leq max_2 \cdot c_1$.

² Lenzerini and Nobili [3] used the notion of *strong satisfiability* for this term.

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

- 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 [2], extend the inequalities based method of [3] to apply to diagrams with class hierarchy constraints, but size of the resulting system of inequalities is exponential in the size of the class diagram.

A method for detection of the cause for non finite satisfiability was first suggested in [3]. The method is based on construction of a directed graph (*digraph*) whose nodes stand for classes and associations, and its edges connect association nodes with their end class nodes. The edges are weighted by the multiplicity constraints, as shown in Figure 2.

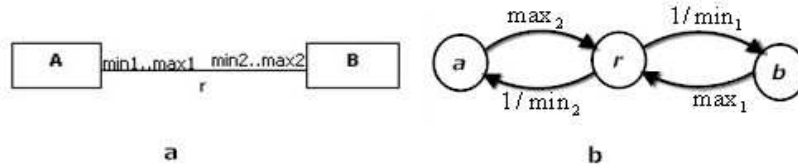


Fig. 2. The digraph representation of a binary association

The *weight of a path* is the product of the weights of its edges. The directed graph is the means for detecting the causes for non finite satisfiability of a class diagram. Cycles whose weight is less than 1 are termed *critical cycles*. They point on non finite satisfiability. Moreover, a critical cycle singles out a non-finitely satisfiable set of multiplicity constraints. Similar approaches are introduced in [4, 11, 12].

3 Efficient Decision of Finite Satisfiability in UML Class Diagrams

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. We present an improved version of the [1] algorithm, which applies to class diagrams with *multiplicity constraints on binary associations*, *class hierarchy constraints*, and *Generalization Set (GS) constraints*.

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.

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 [3] 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 super-class 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 in [6].

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 [3] is applied. Based on Lemma 1, CD is finitely satisfiable if and only if the inequality system of [3] for CD' is solvable. It is not hard to show that this inequality system is equivalent to the inequality system constructed by *FiniteSat*.

The results of this claim can be extended for class diagrams with GS constraints and acyclic class hierarchy structure, or cyclic structure in which class hierarchy cycles do not include *disjoint* or *complete* constraints. The scope of the *FiniteSat* algorithm is defined in the following claim:

Claim 2 (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.

Claim 3 (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.

Table 1 summarizes the results of the above claims.

Graph Structure	With/ Without GS constraints	FiniteSat correctness
Acyclic	Without	correct
	with	correct
Cyclic	Without	correct
	No <i>disjoint</i> or <i>complete</i> in cycles	correct
	<i>disjoint</i> or <i>complete</i> in cycles	sound for unsatisfiability

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

4 Extension of **FiniteSat** to Handle Qualifier Constraints

An association constraint can have an associated *qualifier constraint* that imposes a partitioning on the set of related instances. Figure 4 presents a qualifier constraint that is associated with the *weekDaySchedule* association between the *TV-Network* and the *BroadcastSchedule* classes. The *source* of the qualifier constraint is *TV-Network*, its *target* is *BroadcastSchedule*, its *attribute* is *day*, and its associated multiplicity constraint is 1. The *value domain* of the *day* attribute is *Weekday*. A qualifier constraint might have several attributes, each associated with a value domain. In that case, the *combined value domain* of the qualifier constraint is the cartesian product of the attribute value domains.

The semantics of a qualifier constraint is combined with its associated multiplicity constraint. The general idea is that the combined values of the qualifier attributes impose a partition on the set of target class instances that are linked

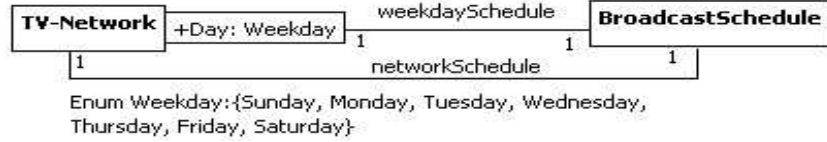


Fig. 4. A qualifier constraint

to a source class instance. That is, for an instance s of the source class, each combined value of the attributes identifies a set of target class instances that are linked to s .

The multiplicity constraint at the target class end, imposes bounds on the size of the partition classes. For example, in Figure 4, the 1 multiplicity constraint states that every day identifies a single broadcast schedule. This semantics raises the problem of non finite qualifier domain values, which might imply non finite partitions of the association, thereby implying non finite associations. Therefore, qualifiers should respect the following syntactic constraint: If the combined value domain of the qualifier is non finite, e.g., *Integer*, the minimum value of the multiplicity constraint at the target class end should be 0, which allows for empty partition classes.

More formally, given a qualifier constraint Q , as described in Figure 5, its semantics is defined as follows: For a legal instance I , Q^I is a function that maps every instance of A^I and a combined value of T_{q_1}, \dots, T_{q_n} to a set of B^I instances:

$$Q^I : A^I \times T_{q_1} \times \dots \times T_{q_n} \rightarrow 2^{B^I}$$

The mapping Q^I satisfies the following constraints:

1. The set of B^I instances to which an A^I instance and a combined domain value are mapped, is restricted by the r multiplicity constraint on the B end:

$$\forall a \in A^I, t_1 \in T_{q_1}, \dots, t_n \in T_{q_n} : \min_2 \leq |Q^I(a, t_1, \dots, t_n)| \leq \max_2$$
2. Q^I is a partition of r^I :
 - (a) The set of B^I instances to which an A^I instance a and a combined domain value are mapped, is included in the set r^I_a of B^I instances that are r^I linked to a :

$$\forall a \in A^I, t_1 \in T_{q_1}, \dots, t_n \in T_{q_n} : Q^I(a, t_1, \dots, t_n) \subseteq r^I_a$$
 - (b) For a given A^I instance, different combined domain values are mapped to disjoint sets of B^I instances:

$$\forall a \in A^I, t_1 \in T_{q_1}, \dots, t_n \in T_{q_n}, t'_1 \in T_{q_1}, \dots, t'_n \in T_{q_n} : Q^I(a, t_1, \dots, t_n) \cap Q^I(a, t'_1, \dots, t'_n) = \emptyset$$

As already commented above, if the combined value domain of a qualifier is non finite, the minimum multiplicity constraint on the target class end must be 0. Otherwise, the non finite value domain implies that the association must be non finite. The reason is that the standard qualifier semantics requires that every

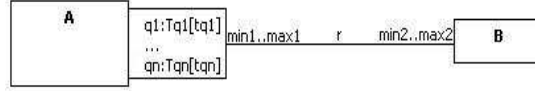


Fig. 5. A general qualifier constraint

value in the combined value domain and every source class instance are mapped to a set of target class instances. We term the standard semantics *universal*.

There is another, more convoluted semantic interpretation for a qualifier constraints, which does not require that every value in the combined values domain is mapped to target class instances. In that semantics, termed *existential* semantics, only values of the combined value domains that are mapped by the qualifier are restricted as above, by the multiplicity constraint and by the association. For example, under the existential semantics, the qualifier value domain in Figure 4 can be non finite, e.g., *Integer*, while the multiplicity constraint on the *BroadcastSchedule* end is 1. The existential semantics is equal to the universal semantics, under a disjunctive multiplicity constraint 0, 1. The extension of *FiniteSat* to apply to qualifier constraints refers to the standard universal semantics.

A qualifier constraint can cause finite satisfiability problems since it tightens the multiplicity constraint of its association. In Figure 4, the qualifier constraint implies that in every legal instance, the number of broadcast schedules, b , is $7 \times t$, where t is the number of TV networks. But, at the same time, $t = b$ by the multiplicity constraints on the *networkSchedule* association. The only solution is that both classes are either empty or infinite. Extending *FiniteSat* to account to qualifier constraints, involves the following addition of step (4):

***FiniteSat* Algorithm - Extension to Qualifier Constraints:**

4 For every qualifier constraint Q , as described in Figure 5:

1. If the combined domain value of Q is non-finite, ignore the qualifier constraint (the cardinality constraint on the other end of the association is handled according to the Lenzerin and Nobili method).
2. Extend the inequality system with the following inequalities:

$$\begin{aligned} \min_1 \times b \leq r \leq \max_1 \times b \\ \min_2 \times a \times t_{q_1} \times \dots \times t_{q_n} \leq r \leq \max_2 \times a \times t_{q_1} \times \dots \times t_{q_n} \end{aligned}$$

Claim 4 *The extension of *FiniteSat* for handling qualifier constraints, properly recognizes finite satisfiability of the class diagram.*

Proof. The proof is based on reduction of finite satisfiability of the given class diagram, CD , to finite satisfiability of a class diagram, CD' , without qualifier constraints. Qualifier constraints with a non finite combined value domain are removed (recall that their minimum multiplicity constraint is 0). A qualifier constraint as in Figure 5 is replaced by two associations and a new class, as in

Figure 6. The instances of the new class Q stand for all combinations of an A instance and a value in the combined domain value of Q . The reduction is correct since the r links in CD are compositions of the r_q and r' links in CD' .

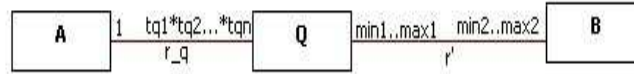


Fig. 6. A reduced qualifier constraint

5 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 . In Figure 7 such a mapping must exist between the extensions of the *Card* class and the *Team-Member* association.

There are two different semantic interpretations for association class hierarchy constraints: *extensional* and *intensional* [13]⁴.

- The *extensional semantics*: A class hierarchy constraint $AC^r \prec AC^q$ enforces a subset constraint between the associated associations r and q . That is, every link of the sub association r is a link of the super association q .
- The *intensional semantics*: A class hierarchy constraint $AC^r \prec AC^q$ requires only that the sub-association r satisfies all the constraints imposed on the super association q .

Unlike the qualifier constraint, finite satisfiability is invariant under the interpretation of association class hierarchy. The reason is that finite satisfiability is sensitive only to multiplicity constraints, and does not query subset relations. Therefore, since both interpretations guarantee the same multiplicity constraints for the sub-association, finite satisfiability is not affected by the concrete selection.

⁴ The logic based semantics that Berardi et al. [5] provide for UML class diagrams seems to overlook the implication of association class hierarchy on their associated associations. Their semantics requires only the 1 : 1 mapping between an association class extension to its associated association extension. Therefore, for $AC^r \prec AC^q$, the mappings $AC^r \leftrightarrow r$ and $AC^q \leftrightarrow q$ might differ, implying that r links might not satisfy the q constraints.

Extending **FiniteSat** to account for association class constraints involves the following addition of step (5), 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:

- 5 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 $AC^r \prec^* AC^q$ ⁵, let r inherit all the multiplicity constraints of q . That is, if q has the multiplicity constraint $q(qa : A_1[\min_{A_1}, \max_{A_1}], qb : B_1[\min_{B_1}, \max_{B_1}])$, and r has the constraint $r(ra : A_2[\min_{A_2}, \max_{A_2}], rb : B_2[\min_{B_2}, \max_{B_2}])$, where the roles qa, qb correspond to the roles ra, rb , respectively, then, add the new multiplicity constraint on r :
 $r(qa : A_2[\min_{A_1}, \max_{A_1}], qb : B_2[\min_{B_1}, \max_{B_1}])$,
and apply the Lenzerini and Nobili construction to the new constraint.
 3. Apply step (5.2) transitively, to all sub association classes of AC^q .

Example 1. Consider the class diagram in Figure 7.a. The subset constraint between the *VIPContract* and the *Contract* association classes tightens the multiplicity constraint of *Member* in *VIPContract* into 2..3, which causes a finite satisfiability problem. Applying the extended **FiniteSat** algorithm yields the unsolvable inequality system below, implying that the class diagram is not finitely satisfiable.

$$t \geq s, \quad t \geq l, \quad c = m, \quad t \leq c \leq 3t, \quad c \geq v,$$

$$v = m, \quad 2l \leq v \leq 6l, \quad ma = m, \quad ma = 4l.$$

Claim 5 (Correctness of the association class extended **FiniteSat)** *The extension of **FiniteSat** for handling association class constraints, properly recognizes finite satisfiability of the class diagram.*

Proof. (Sketched) The proof is based on two successive reductions of finite satisfiability, from a class diagram CD into a simpler class diagram CD' (Lemma 2), and from CD' into the solvability of an inequality system (Lemma 3).

Lemma 2. *Finite satisfiability of a class diagram CD with association classes can be reduced into finite satisfiability of a class diagram CD' that replaces class hierarchy constraints by regular associations (as in Lemma 1).*

Proof. CD' is a class diagram without class hierarchy constraints, but still with association classes. The reduction is proved using a mapping T , that maps a legal instance I' of CD' into a legal instance I of CD . The mapping differs according to the selected semantic interpretation of association class hierarchy. For the

⁵ \prec^* is the transitive closure of the relation \prec .

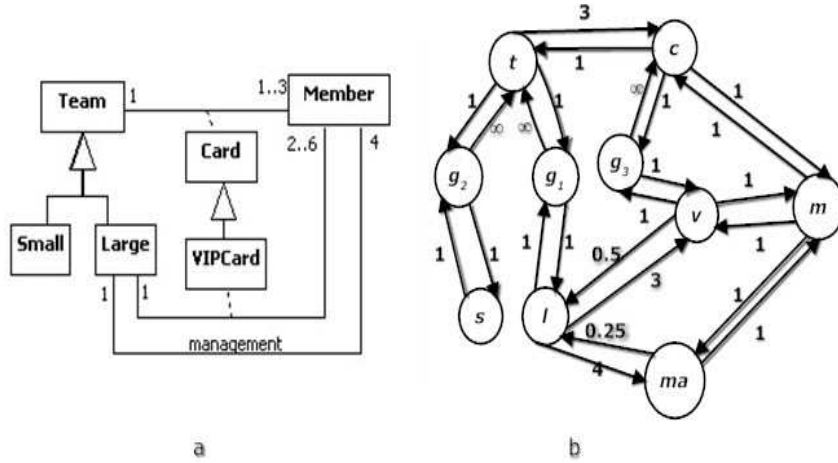


Fig. 7. An association class hierarchy constraint with its digraph translation

extensional semantics, the mapping T collapses links that are *ISA* related (via their associated association class instances) in I' into a single link in I . For the intensional semantics the mapping T is defined as in Lemma 1.

Lemma 3. *A class diagram with association classes, but with no class hierarchy constraints is finitely satisfiable, if and only if the inequality system that is constructed by the [3] method, with an equality $AC^r = r$ added for every association class AC^r , is solvable.*

Proof. Revision of the [3] proof, by connecting every constructed AC^r instance to a single constructed r link.

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 hierarchy constraint requires going over the whole hierarchy.

Theorem 1 (*FiniteSat* correctness). *For a class diagram CD with binary multiplicity constraints, class hierarchy constraints, *GS* constraints, qualifier 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.*

Deciding Finite Satisfiability in Description Logics: *Finite satisfiability* in Description Logics (as in UML class diagrams) is known to be hard. Standard DL reasoners do not reason about finite satisfiability. In [14] we suggest to efficiently decide finite satisfiability in description logics via a translation into UML class diagrams. For that purpose, we have developed a simple translation of atomic, primitive knowledge bases of popular description logics, into UML class diagrams.

6 Cause Detection of Non Finite Satisfiability of Class Diagram with Unconstrained GSs

In this section we present a cause detection method for non-finite satisfiability in class diagrams with unconstrained GSs, qualified associations and association class constraints. Our method extends the critical cycle based detection method of [3]: It is based on construction of a directed graph (*digraph*), and detection of critical cycles.

Algorithm 2 *The CriticalCycle Algorithm*

Input: A class diagram CD with binary multiplicity constraints, class hierarchy constraints, qualified association and association class constraints.

Output: A directed graph, with weighted edges.

Method:

1. Construct a class diagram CD' as in the correctness proofs for the **FiniteSat** algorithm (replacing a class hierarchy constraint $B \prec A$ by a regular association $isa(rn_1 : B[0, 1], rn_2 : A[1, 1])$, and replacing a qualified association by a regular association as in step 4 of **FiniteSat** (see Section 4).
2. Construct a directed graph $G_{CD'}$ by applying the [3] method to CD' .
3. For every association class AC^r :
 - (a) Unify the nodes of r and AC^r in $G_{CD'}$.
 - (b) Insert to $G_{CD'}$ weighted edges as follow:
 - i. Let $\{r(rn_{i1} : A[\min_{i1}, \max_{i1}], rn_{i2} : B[\min_{i2}, \max_{i2}]) \mid 1 \leq i \leq n\}$ be the set of all multiplicity constraints imposed on r by applying step 5.2 of **FiniteSat** (Section 5). Define $\min_1 = \max\{\min_{i1} \mid 1 \leq i \leq n\}$, $\min_2 = \max\{\min_{i2} \mid 1 \leq i \leq n\}$, $\max_1 = \min\{\max_{i1} \mid 1 \leq i \leq n\}$ and $\max_2 = \min\{\max_{i2} \mid 1 \leq i \leq n\}$.
 - ii. Apply the [3] method to $r(ra : A[\min_1, \max_1], rb : B[\min_2, \max_2])$.

Claim 7 A critical cycle in $G_{CD'}$ corresponds to a set of not finitely satisfiable constraints, i.e., at least one class affected by these constraints is not finitely satisfiable.

Proof. (Sketch) The proof follows from the correctness proof of Algorithm **FiniteSat** (Theorem 1), and from [3].

Conclusion: A class diagram with multiplicity constraints on binary associations, unconstrained GS constraints, qualifier constraints and association class constraints is finitely satisfiable if and only if the digraph constructed by the *CriticalCycle* algorithm does not include critical cycles.

Going back to Figure 7-b, the critical cycle $\langle ma, l, v, m, ma \rangle$ singles out a non-finitely satisfiable sub-diagram of Figure 7-a (the non-finite satisfiability of this class diagram was already recognized by the *Finite Sat* algorithm).

7 Future Work

In this paper, we extended the scope of *FiniteSat* algorithm to handle also qualifier and association class constraints. Furthermore, we also introduced a method for detecting the cause of finite satisfiability problems in class diagrams with unconstrained GSs, qualified associations and association class constraints.

In the future, we plan to further extend the scope of the *FiniteSat* and of the cause detection algorithms. Another direction involves the possibility of expanding our method with heuristics for repairing finite satisfiability problems, following the ideas of [12, 15]. An implementation of the *FiniteSat* algorithm is now being developed. We intend to use it for testing finite satisfiability of large class diagrams like the UML meta-model or the Java library classes.

References

- [1] Maraee, A., Balaban, M.: Efficient reasoning about finite satisfiability of uml class diagrams with constrained generalization sets. In: The 3rd European Conference on Model-Driven Architecture, Springer (2007) 17–31
- [2] Calvanese, D., Lenzerini, M.: On the interaction between isa and cardinality constraints. In: The 10th IEEE Int. Conf. on Data Engineering, Washington, DC, USA, IEEE Computer Society (1994) 204–213
- [3] Lenzerini, M., Nobili, P.: On the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems* **15(4)** (1990) 453–461
- [4] Thalheim, B.: *Entity Relationship Modeling*, Foundation of Database Technology. Springer-Verlag (2000)
- [5] Berardi, D., Calvanese, D., Giacomo, D.: Reasoning on uml class diagrams. *Artificial Intelligence* **168** (2005) 70–118
- [6] Maraee, A.: Efficient methods for solving finite satisfiability problems in uml class diagrams. Master’s thesis, Ben-Gurion Univ., Israel (2007)
- [7] Artale, A., Calvanese, D., Kontchakov, R., Ryzhikov, V., Zakharyashev, M.: Complexity of reasoning in entity relationship models. In: 20th International Workshop on Description Logics (DL-2007). (2007)
- [8] Lutz, C., Sattler, U., Tendera, L.: The complexity of finite model reasoning in description logics. *Inf. Comput.* **199** (2005) 132–171
- [9] Schild, A.: A correspondence theory for terminological logics: preliminary report. Technical Report KIT-BACK, FR 5-12 (1991)

- [10] Thalheim, B.: Fundamentals of cardinality constraints. In: The 11th International Conference on the Entity-Relationship Approach, London, UK, Springer-Verlag (1992) 7–23
- [11] Hartmann, S.: Graph-theoretical methods to construct entity-relationship databases. In: The 21st International Workshop on Graph-Theoretic Concepts in Computer Science, London, UK, Springer-Verlag (1995) 131–145
- [12] Hartmann, S.: Coping with inconsistent constraint specifications. In: The 20th International Conference on Conceptual Modeling, London, UK, Springer-Verlag (2001) 241–255
- [13] OMG: The Unified Modeling Language (OMG UML), Superstructure, V2.1.2. (2006)
- [14] Maraee, A., Balaban, M.: A uml-based method for deciding finite satisfiability in description logics. In: The 21st International Workshop on Description Logics, DL2008, Dresden, Germany (2008)
- [15] Hartmann, S.: Soft constraints and heuristic constraint correction in entity-relationship modelling. In: Semantics in Databases, Springer (2001) 82–99