# Efficient Reasoning about Finite Satisfiability of UML Class Diagrams with Constrained Generalization Sets

Azzam Maraee[1] and Mira Balaban[2] *

[1] Information Systems Engineering Department
[2] Computer Science Department
Ben-Gurion University of the Negev, Beer-Sheva 84105, ISRAEL.
mari@bgu.ac.il, mira@cs.bgu.ac.il

**Abstract.** UML class diagrams play a central role in the design and specification of software, databases and ontologies. The model driven architecture approach emphasizes the central role that models play, towards achieving reliable software. It is important that models are correct and that problems are detected as early as possible in the software design process. However, current case tools do not support reasoning tasks about class diagrams and enable the construction of erroneous models. There is an urgent need for methods for detecting analysis and design problems. In this paper, we present a linear programming based method for reasoning about finite satisfiability of UML class diagrams with constrained generalization sets. The method is simple and efficient and can be easily added to a case tool. It improves over existing methods that require exponential resources and extends them to new elements of class diagrams.

**Keywords:**
UML class diagram, finite satisfiability, consistency, cardinality constraints, reasoning about class diagram, generalization set constraints, class hierarchy structure.

## 1 Introduction

The Unified Modeling Language (UML) is nowadays the industry standard modeling framework, including multiple visual modeling diagrams collectively, referred to as a UML model. Traditionally, UML models are used for analysis and design of complex systems. Their relevance has increased with the advent of the Model-Driven Development (MDD) approach, in which analysis and design models play an essential role in the process of software development. Recently, with the emergence of web-enabled agent technology, UML models are used also for ontology representation, and construction and extraction of ontologies [7].

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

In view of their wide popularity, it is highly important that UML models provide reliable support for the designed systems, and be subject to stringent quality assurance and quality control criteria [21]. Indeed, an extensive amount of research efforts is devoted to formalization of UML models, specification of their semantics, and development of reasoning and correctness checking methods [2, 16]. Moreover, with the prevalence of the Model Driven Engineering approach, it is expected that all information in a design model will be effective in its successive models.

Modeling problems usually arise when models are scaled to model large, distributed applications. A model may originate from different sources and a large number of designers can be involved in the modeling process. Designers are highly prone to making mistakes, and combining information from different sources gives rise to potential conflicts [3, 6, 10]. [14] shows that defects often remain undetected, even if the model is read attentively by practitioners.

It is highly important that models are tested for correctness, and that problems are detected as early as possible in the software design process. Nevertheless, current case tools do not support reasoning about UML models, and enable the construction of erroneous ones. Furthermore, implementation languages still do not enforce design level constraints. Hence, there is an urgent need for reasoning methods for detecting analysis and design problems.

*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 systems structure in terms of classes, associations, and constraints imposed on classes and their inter-relationships. Constraints provide an essential means of knowledge engineering, since they extend the expressivity of diagrams. UML supports class diagram constraints such as cardinality constraints, class hierarchy constraints, and inter-association constraints. Example 1 below, presents a class diagram that includes cardinality and hierarchy constraints.

*Example 1.* Figure 1 presents a class diagram with three classes named *Academic*, *Graduate* and *FacultyMember*, one association *advisor-student* between instances of the *Academic* and the *Graduate* classes, with roles named *advisor* and *student*, respectively, a cardinality constraint that is imposed on this association, and a generalization set with a super-class *Academic* and sub-classes *Graduate* and *FacultyMember*. The cardinality constraint states that every *Graduate* student must be advised by exactly one *Academic*, while every *Academic* must advise exactly two *Graduate* students. The generalization set states that *Graduates* and *FacultyMembers* are *Academic* as well, implying that the *advisor* of a *Graduate* can be a *Graduate* or a *FacultyMember* or another *Academic*.

In the presence of constraints a class diagram may turn inconsistent, as it might impose constraints that cannot be finitely satisfied. Figure 1, presents a multiplicity constraint cycle that involves a compound class, *Graduate*, whose instances must be related to *Academic* instances. Therefore, the number of *student-advisor* links in every diagram instance must be both, $G \cdot 1$ and $A \cdot 2$, assuming

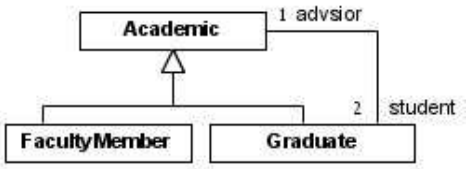


**Fig. 1.** A Class Diagram with a Finite Satisfiability Problem

that $G$ and $A$ are the number of graduates and academics, respectively. Therefore, the extensions of *Graduate* and *Academic* must satisfy $G = A \cdot 2$, while the *Graduate* extension is a subset of the *Academic* extension. This constraint can be satisfied only by empty or infinite extensions. Such problems are termed *finite satisfiability* problems.

The problem of finite satisfiability has been studied in the context of various kinds of conceptual schemata [1, 3, 5, 8, 10, 15, 20]. There are methods for testing finite satisfiability, for detecting causes for unsatisfiability, and for heuristic suggestions for diagram correction. Yet, no method provides a feasible solution for detecting lack of finite satisfiability for the combination of cardinality constraints, class hierarchy constraints, and generalization sets constraints.

In this paper, we present a linear programming based method for reasoning about finite satisfiability of UML class diagrams with constrained generalization sets. The method is based on a reduction to the algorithm of Lenzerini and Nobili [15] that was applied only to ER-diagrams without class hierarchies. It is simple and feasible since it adds in the worst case only a linear amount of entities to the original diagram. It improves over previous extensions of the Lenzerini and Nobili method that require the addition of an exponential number of new entities to the original diagram [5]. An implementation of our method within a UML case tool is currently under development.

The paper is organized as follows: Section 2 presents the finite satisfiability notion, summarizes relevant methods for detecting finite satisfiability problems in class diagrams, introduces the Generalization Set notion of UML2.0, and classifies different class hierarchy structures. Although this paper focuses only on finite satisfiability problems, for the sake of completeness we also introduce the consistency notion. Section 3 describes a polynomial time algorithm for testing finite satisfiability of UML class diagrams with unconstrained generalization sets. Section 4 extends the algorithm to operate on constrained generalization sets, and investigates the limits of this method. Section 5 is the conclusion and discussion of future work.

## 2 Background

The standard set theoretic semantics of class diagrams associates a class diagram with *class diagram instances* in which classes have extensions that are sets of

objects that share structure and operations, and associations have extensions that are relationships among class extensions. We denote class symbols as $C$, association symbols as $A$, and role symbols as $rn$. Henceforth, we shorten expressions like "instance of an extension of $C$" by "instance of $C$" and "instance of an extension of $A$" by "instance of $A$".

A cardinality constraint (also termed multiplicity constraint) imposed on a binary association $A$ between classes $C_1$ and $C_2$ with roles $rn_1$, $rn_2$, respectively, is symbolically denoted:

$$A(rn_1 : C_1[\ min_1,\ max_1], rn_2 : C_2[\ min_2,\ max_2]) \tag{1}$$

The multiplicity constraint $[min_1, max_1]$ that is visually written on the $rn_1$ end of the association line is actually a participation constraint on instances of $C_2$. It states that an instance of $C_2$ can be related via $A$ to $n$ instances of $C_1$, where $n$ lies in the interval $[min_1, max_1]$. A class hierarchy constraint between a super class $C_1$ and a subclass $C_2$ is written $ISA(C_2, C_1)$ and called also *ISA constraint*. It states a subset relation between extensions of $C_2$ and $C_1$.

A *legal instance* of a class diagram is an instance where the class and association extensions satisfy all constraints in the diagram. Correctness of a class diagram involves consistency and satisfiability notions, that are discussed in [2, 5, 15, 20]. We further elaborate this terminology, and suggest additional notions, in order to facilitate a more accurate definition of correctness.

- **Consistency Notions**:
  1. A ***class diagram is consistent (satisfiable)*** if it has an instance with at-least one non-empty class extension. Otherwise, it is inconsistent.
  2. A ***class C in a class diagram is consistent*** if there is an instance $I$ in which the extension of $C$ is non-empty ($C$ is said to be *consistent in* I). Otherwise, it is *inconsistent, (unsatisfiable)*.
  3. A ***class diagram is all class consistent*** if every class is consistent.
  4. A ***class diagram is fully consistent*** if it has an instance in which all classes are consistent.
- **Finite Satisfiability Notions**:
  1. A ***class is finitely satisfiable*** in a class diagram if there is a finite instance in which the class is consistent (A class diagram instance is finite if all class extensions are finite).
  2. A ***class diagram is all class finitely satisfiable*** if for every class there is a finite instance in which the class is consistent. Lenzerini and Nobili [15] used the notion of *strong satisfiability* for this term.
  3. A ***class diagram is fully finitely satisfiable*** if it has a finite instance in which all classes are consistent.

The important notions for consistency and finite satisfiability are those of *full consistency* and *full finite satisfiability*. [17] shows that full consistency is equivalent to all class consistency, and full finite satisfiability is equivalent to all class finite satisfiability. Inconsistency and lack of finite satisfiability are errors in design that might delay system development and increase its cost [13]. The first

because an inconsistent class diagram does not have a non-empty extension, and the latter because there is no finite and non-empty extension [4]. The consistency problem is instigated in [2, 12].

### 2.1 Methods for Reasoning about Finite Satisfiability of UML Class Diagrams

The method of Lenzerini and Nobily is defined for *Entity-Relationship* (*ER*) diagrams that include *Entity Types* (*Classes*), *Binary Relationships* (*Binary Associations*), and *Cardinality Constraints*. The method consists of a transformation of the cardinality constraints into a set of linear inequalities whose size is polynomial in the size of the diagram. All class finite satisfiability of the ER diagram reduces to solution existence of the associated linear inequalities system. The linear inequalities system is defined as follow:

1. For each association $R(rn_1 : C_1[min_1, max_1], rn_2 : C_2[min_2, max_2])$ insert the following inequalities:
   - 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 $max_1 \neq * :\ r \leq max_1 \cdot c_2$.
2. For every entity or association symbol $T$ insert the inequality: $T > 0$.

Lenzerini and Nobili also present a method for identification of causes for non-satisfiability. This method is based on a transformation of the conceptual schema into a graph and identification of critical cycles. Similar approaches are introduced in [20, 8]. Hartman, in [9] further develops methods for handling finite satisfiability problems in the context of database key and functional dependency constraints. Heuristic methods for constraint corrections are presented in [10, 11].

Calvanese and Lenzerini, in [5], extend the inequalities based method of Lenzerini and Nobili [15] to apply to schemata with class hierarchy constraints. The expansion is based on the assumption that class extensions may overlap. They provide a two stage algorithm in which the finite satisfiability problem of a class diagram with *ISA* constraints is reduced into the finite satisfiability problem of a class diagram without *ISA* constraints. Then, similarly to [15], they check all class finite satisfiability of the new class diagram by deriving a special system of linear inequalities (different from that of [15]).

The class diagram transformation process of [5] is fairly complex, and might introduce, in the worst case, an exponential number, in terms of the input diagram size, of new classes and associations. The method was further simplified in [4], where class overlapping is restricted to class hierarchy alone. The simplification of [4] reduces the overall number of new classes and associations, but the worst case is still exponential. Example 2 presents the application of [4] to Figure 1.

*Example 2.* The application of the [4] method yields four classes and eight associations. Each class and association is represented by a variable in the resulting inequalities system. The variables are:

1. **Class variables:** $a_1$ for an *Academic* that is neither a *Graduate* nor a *FacultyMember*; $a_2$ for an *Academic* that is a *Graduate* but not a *FacultyMember*; $a_3$ for an *Academic* that is a *FacultyMember* but not a *Graduate*; $a_4$ for an *Academic* that is simultaneously a *Graduate* and a *FacultyMember*.
2. **Association variables:** $\{ad_{ij} | 1 \leq i \leq 4 \wedge j \in \{2,4\}\}$. Every specialized association relates two new classes, one for the advisor role and the other for the *student* role. The indexes represent the indexes of the class variables. For example, the variable $r_{12}$ represents the specialization of the *advisor-student* association to an association between *Academics* who are neither *Graduates* nor *FacultyMembers* (the $a_1$ variable) and *Academics* specialized to *Graduates* but not to *FacultyMembers* (the $a_2$ variable).

The inequalities system below results from application of the method of [4] to Figure 1. Equations 1-4 translate the 2..2 multiplicity, equations 5-6 translate the 1..1 multiplicity, and the inequalities in 7-9 represent the satisfiability conditions. The inequalities system is unsolvable, implying that the class diagram in Figure 1 is finitely unsatisfiable.

1. $2a_1 = ad_{12} + ad_{14}$.
2. $2a_2 = ad_{22} + ad_{24}$.
3. $2a_3 = ad_{32} + ad_{34}$.
4. $2a_4 = ad_{42} + ad_{44}$ .
5. $a_2 = ad_{12} + ad_{22} + ad_{32} + ad_{42}$.
6. $a_4 = ad_{14} + ad_{24} + ad_{34} + ad_{44}$.
7. $a_1, a_2, a_3, a_4, ad_{12}, ad_{14}, ad_{22}, ad_{24}, ad_{32}, ad_{34}, ad_{42}, ad_{44} \geq 0$.
8. $a_1 + a_2 + a_3 + a_4 > 0$.
9. $ad_{12} + ad_{14} + ad_{22} + ad_{24} + ad_{32} + ad_{34} + ad_{42} + ad_{44} > 0$.

### 2.2 UML2.0 Class Hierarchy Concepts: Generalization Sets

In UML2.0 class hierarchy constraints are expressed using the *Generalization Set* (*GS*) concept, which is similar to the former class hierarchy grouping construct [18]. A *GS*s includes a superclass and a set of sub classes (different from the super class). The semantics is that the sub classes denote sub sets of the set denoted by the super class. *GS*s may be constrained as follows [18, 19]:

1. *complete* - An instance of the superclass is an instance of at least one subclass.
2. *incomplete*- There might be instances of the superclass of that are not instances of any subclass.
3. *disjoint*- Subclasses are mutually exclusive.
4. *overlapping* - Subclasses may overlap.

The *GS* constraints can be combined to form one of the following valid combination: {complete, disjoint}, {incomplete, disjoint}, {complete, overlapping}, {incomplete, overlapping}. Figure 2 shows a *disjoint* constraint.

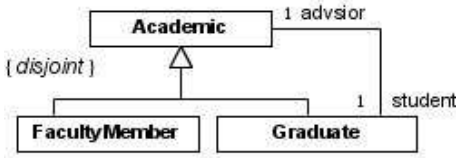


**Fig. 2.** Constrained Generalization Set

### 2.3 Classification of Class Hierarchy Structures

Class hierarchy can arise in various structures that affect the finite satisfiability decision algorithm. We distinguish three parameters that determine the class hierarchy structures and content:

1. **ISA Graph Structure**: *ISA* constraints can form three kinds of graph structures:
   (a) **Tree Structure**, as in Figure 1: A subclass has a single super class.
   (b) **Acyclic Structure**: Multiple inheritance is allowed, but the undirected induced subgraph formed by the *ISA* constraints is acyclic. For example, in Figure 3-a, the hierarchy structure is not a tree, as $F$ is a sub class of both $C$ and $D$, but the undirected class hierarchy graph is acyclic. The acyclic structure prevents multiple inheritance with a common ancestor-class.

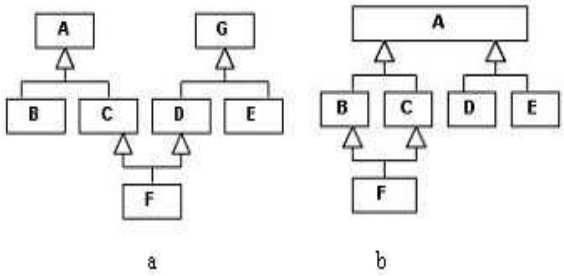


**Fig. 3.** Unconstrained Hierarchy Structures

   (c) **Graph Structure**, as in Figure 3-b: unrestricted multiple inheritance.
2. **Presence of *GS* Constraints**.
3. **Number of *GS*s per superclass**, as in Figure 3-b.

We use an abbreviated notation that specifies the value of these parameters. The *hierarchy structure* is denoted by one of {T,A,G}, standing for *Tree structure, Acyclic graph*, and *Graph*, respectively. The presence of *GS* constraints is denoted by *C*, and the presence of multiple *GS*s per superclass is denoted by *M*.

The multiple *GS*s per superclass distinction is relevant only for tree structure hierarchies. For acyclic on graph hierarchies, multiple *GS*s per a single superclass are allowed by the graphical structure. The resulting hierarchy variants are: [T]-GS for tree structured unconstrained hierarchy with a single *GS* per superclass; [T-C]-GS for tree structured constrained *GS*s with a single *GS* per superclass; [T-M]-GS for tree structured unconstrained hierarchy with multiple *GS*s per super class; [T-C-M]-GS for a constrained tree hierarchy with multiple *GS*s per super class; [A]-GS for an unconstrained acyclic hierarchy; [A-C]-GS for a constrained acyclic hierarchy; [G]-GS for unconstrained graph hierarchy; [G-C]-GS for a constrained graph hierarchy.

## 3 Reasoning about Finite Satisfiability of UML Class Diagrams with Unconstrained *GS*s

In this section, we present a method for reasoning about finite satisfiability of UML class diagrams with unconstrained *GS*s. We start with a tree structured hierarchy [T]-GS, and extend it to the hierarchical structures: {[T-M],[A],[G]}-GS.

The method builds on top of the Lenzerini and Nobili [15] algorithm described in Section 2. We reduce the *finite satisfiability* problem of a class diagram with *ISA* constraints, into the *finite satisfiability* problem of a class diagram that is handled by [15]. First, we state that *all class finite satisfiability* implies a *full finite satisfiability* The proof is similar to the proof for the case of the restricted ER diagrams, as presented in [15].

**Theorem 1.** *If a class diagram is all class finitely satisfiable then it is fully finitely satisfiable.*

*Proof.* (Sketched) The theorem is proved by the following argument: Every two disjoint instances can be combined into a single instance of the class diagram. The argument holds due to the special character of UML class diagram constraints, which are closed under disjoint instance combination. For full proof consult [17].

### 3.1 Testing the Finite Satisfiability of Class Diagrams with [T]-GS

**Algorithm 1**:

- **Input**: A class diagram *CD* that includes binary associations and [T]-GS.
- **Output**: True, if *CD* is *all class finitely satisfiable*; false otherwise.
- **Method**:
  1. Class diagram reduction - Create a new class diagram $CD'$ as follows:
     (a) Initialize $CD'$ by the input class diagram $CD$.
     (b) Remove from $CD'$ all generalization set constructs.
     (c) For every removed generalization set construct create new binary associations between the superclass to the subclasses, with 1..1 participation constraint for the subclass (written on the super class edge in the diagram) and 0..1 participation constraint for the super class.

2. Apply the Lenzerini and Nobili algorithm to $CD'$.

*Example 3.* Figure 4 is the reduced class diagram of Figure 1, following step 1 in the algorithm. Applying the inequalities method of [15] (step 2) yields the inequalities system below. We use the symbols $ad$ for *Academic*, $g$ for *Graduate*, $fm$ for *FacultyMember*, $as$ for the *advisor-student* association, and $isa_1$, $isa_2$ for the new associations $ISA_1$ and $ISA_2$ respectively.

$$as \geq 2ad,\ as \leq 2ad,\ as \leq g,\ as \geq g,\ isa_1 \geq g,\ isa_1 \leq g,\ isa_1 \leq ad,\ isa_2 \geq fm,$$
$$isa_2 \leq fm,\ isa_2 \leq ad, \text{ and}$$
$$as > 0\ ,ad > 0,\ \ g > 0,\ \ fm > 0,\ \ isa_1 > 0,\ \ isa_2 > 0$$

This system has no solution and therefore the [15] algorithm returns False. The same result was obtained in Section 2 by applying the [5],[4] algorithm to Figure 1.

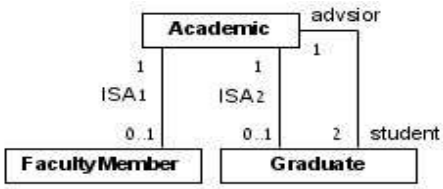


**Fig. 4.** The Reduced Class Diagram of Figure 1

**Claim 1: [Correctness of Algorithm 1]** Algorithm 1 tests for *all class finite satisfiability* of class diagrams with [T]-GS.

*Proof.* (sketched) the claim builds on showing that the translated class diagram $CD'$ preserves the satisfiability of the input class diagram $CD$. Full proof appears in [17].

**Claim 2: [Complexity of Algorithm 1]** Algorithm 1 adds to the [15] method an $O(n)$ time complexity, where $n$ is the size of the class diagram (including associations, classes and ISA constraints).

*Proof.* The additional work involves the class diagram reduction, which creates a class diagram with the same set of classes and one additional association that replaces every class hierarchy constraint. Since there is a linear additional work per ISA constraint, the overall additional work is a linear to the size of the class diagram.

### 3.2 Extensions for {[T-M], [A], [G]}-GS

Algorithm 1 applies properly to the other unconstrained structures: {[T-M], [A], [G]}-GS. The extensions preserve the correctness of the algorithm since the reduction of *all class finite satisfiability* of $CD$ to that of $CD'$ is still correct. The more complex structure does not break the reduction because as long as the *GS*s are not constrained, *ISA* constraints can be simulated by regular links between the involved classes. Different instances of a superclass $C$ in $CD'$ can be unified into a single instance of $C$ in $CD$, without breaking any constraints.

## 4 Reasoning about Finite Satisfiability of UML Class Diagrams with Constrained Generalization Sets

Adding *GS*-constraints to the class diagram imposes additional requirements on its finite satisfiability problem. In order to test finite satisfiability under *GS*-constraints, the finite satisfiability problem of a class diagram $CD$ with *ISA* constraints, is reduced into the finite satisfiability problem of a "constrained" class diagram $CD'$ without class hierarchy. The additional constraints on $CD'$ preserve the constraints on the *GS*s of $CD$. The inequalities system obtained by applying the method of [15] to $CD'$ is expanded with new inequalities that reflect the *GS* constraints. This algorithm is first introduced for *single GS*-constraints, i.e., the four *GS*-constraints *disjoint, complete, incomplete, overlapping*, and then expanded for handling combinations of *pair GS*-constraints. We begin with an algorithm for deciding finite satisfiability of tree structured ([T-C]-GS) class diagrams. We show that the algorithm applies also to [T-C-M]-GS and to [A-C]-GS class diagrams. Finally we explore the limits of the algorithm for the [G-C]-GS class diagrams. We show that for graph structured class hierarchies, the algorithm can handle the *overlapping* and the *incomplete GS*-constraints, but falls short for deciding finite satisfiability for the *disjoint* and the *complete GS*-constraints.

The *incomplete* and the *overlapping* constraints of generalization sets have a "possibilistic" semantics: The first states that there might be direct instances of the superclass, and the second states that subclasses may overlap. Finite satisfiability for these constraints requires the realization of the possibilistic nature. That is, the *incomplete* constraint requires the existence of direct instances of the superclass, and the *overlapping* constraint requires the existence of common instances for subclasses. For the finite satisfiability problem, we require the existence of an instance in which incomplete super-classes have direct instances, and overlapping subclasses have common instances.

### 4.1 Testing Finite Satisfiability of Tree Structured ([T-C]-GS) Class Diagrams

**Algorithm 2**:

- **Input**: A class diagram $CD$ that includes binary associations and [T-C]-GS.

– **Output**: True, if $CD$ is *all class finitely satisfiable*; false otherwise.
– **Method**:
1. Class diagram reduction:
   (a) Steps 1.a, 1.b, 1.c from Algorithm 1.
   (b) For every generalization set $C, C_1, ..., C_n$ in $CD$, add constraint *Const* on its classes as follows:
   for *disjoint/overlapping* constraint, *Const* is: "there is no/(at least one) instance of class $C$ which is associated with more than one instance from $C_1$,... $C_n$ via the $ISA$ links";
   for *complete/incomplete* constraint, *Const* is: "all/part of the instances of class $C$ are associated with the instances of the classes $C_1, ..., C_n$ via the $ISA$ links".
2. Inequalities system construction:
   (a) Create the inequalities system for $CD'$ according to the Lenzerini and Nobili algorithm.
   (b) For every single constraint *Const* added in step 1b, extend the inequalities system, as follows:
      i. *Const* = *disjoint*: $C \geq \sum_{j=1}^{n} C_j$.
      ii. *Const* = *complete*: $C \leq \sum_{j=1}^{n} C_j$.
      iii. *Const* = *incomplete*: $\forall j \in [1, n].C > C_j$.
      iv. *Const* = *overlapping*: Without inequality.
   (c) For every pair of constraints added in step 1b, extend the inequalities system, as follows:
      i. *disjoint, incomplete*: $C > \sum_{j=1}^{n} C_j$.
      ii. *disjoint, complete*: $C = \sum_{j=1}^{n} C_j$.
      iii. *overlapping, complete*: $C < \sum_{j=1}^{n} C_j$.
      iv. *overlapping, incomplete*: $\forall j \in [1, n].C > C_j$.
3. Apply the Lenzerini and Nobili algorithm to $CD'$.

*Example 4.* Consider Figure 2. The interaction between the cardinality constraint, the hierarchy, and the $GS$ constraints causes a finite satisfiability problem. Applying the method of [15] with the extension in Algorithm 2, step 2.b.i, to the reduced class diagram of Figure 2 yields the unsolvable inequalities system (same variables from Example 3) presented below, implying that the class diagram is finitely unsatisfiable.

$$isa_1 = g,\ isa_1 \leq ad,\ isa_2 = fm\ ,isa_2 \leq ad,\ as = ad,\ ad = g\ ,\ ad > 0,\ as > 0,$$
$$g > 0, fm > 0,\ isa_1 > 0,\ isa_2 > 0, \text{ and the } \textit{disjoint} \text{ inequality: } ad \geq g + fm$$

**Comment**: The inequalities that are used in step (2.b) for satisfying the single $GS$-constraints are not mutually exclusive. Indeed, there are solutions for the inequalities system that can imply finite satisfiability for several constraints. For example, a solution that yields equality in a *disjoint* inequality implies that the *disjoint* constraint can be replaced by a *complete* constraint, without affecting finite satisfiability, and vice versa. Step (2.c) handles pairs of $GS$- constraints

that result from combinations of *disjoint / overlapping* with *complete / incomplete*. The single constraint inequalities are tightened so to meet the combined constraints.
**Claim 3: [Correctness of Algorithm 2]** Algorithm 2 tests for *all class finite satisfiability* of class diagrams with [T-C]-GS hierarchy structure.

*Proof.* (Sketched) The claim builds on showing that the translated class diagram $CD'$ together with its associated constraints, preserves the *all class finite satisfiability* of the input class diagram $CD$. As for the second step of the algorithm, we show that for each constraint the additional inequality (or equality) provides a necessary and sufficient condition for the existence of a $CD'$ instance that satisfies the generalization set constraint. For example, inequality [i] in step 2.b of Algorithm 2 characterizes the existence of a $CD'$ instance that satisfies the referenced *disjoint* constraint. For full proof consult [17].

**Claim 4: [Complexity of Algorithm 2]** Algorithm 2 adds an $O(n)$ time complexity to the [15] method, where $n$ is the size of the class diagram (including associations, classes and ISA constraints).

*Proof.* The additional work involves the class diagram reduction, which creates a class diagram with the same set of classes and one additional association that replaces every class hierarchy constraint. In addition, every *GS* constraint adds a single inequality. Since the work per generalization set is linear in its size, the overall additional work is linear in the size of the class diagram.

The inequalities of the pair *GS*-constraints are not exclusive. The first and second inequalities imply, each, the last. Therefore, finite satisfiability with the pair constraints {*disjoint, incomplete*}/ {*disjoint, complete*} implies finite satisfiability with the {*overlapping, incomplete*} constraints. This observation leads to the following conclusion:
**Conclusion:** If a tree structured class diagram $CD$ is fully finitely satisfiable, then a class diagram $CD'$ which is obtained from $CD$ by replacing pairs of *GC*-constraints {*disjoint, incomplete*}/ {*disjoint, complete*} with {*overlapping, incomplete*} is also fully finitely satisfiable.

### 4.2 Extension of Algorithm 2 to {[T-C-M], [A-C], [G-C]}-GS Hierarchy Structure - Exploring the Limits of the Suggested Method

Algorithm 2 extends properly to the {[T-C-M], [A-C]}-GS hierarchy structures, but it does not extend to the full case of [G-C]-GS hierarchies. The single *GS*-constraints *incomplete* and *overlapping* cause no problems. But the presence of the *disjoint* or the *complete* constraints within cyclic class hierarchies fails the algorithm. The reason is that in general graph structured class hierarchies, these *GS*-constraints have an implicit global effect on other generalization sets in a cycle. We now demonstrate the problems, and explain why the method of Algorithm 2 cannot handle these cases.

**Presence of a *disjoint* GS-Constraint in a [G-C]-GS class diagram**: Consider the class diagram in Figure 5-a. The *disjoint* constraint imposed on the generalization set $\{A,\ B,\ C, D\}$ implies that in every instance, the extension of $E$ properly includes the extension of $D$. But, object members of class $E$ are mapped in a 1:1 manner to members of $D$, implying that the sets have the same size. The only solution for proper set inclusion with equal size is that the sets are either empty or infinite. Therefore, the diagram is not fully finitely satisfiable.

Nevertheless, Algorithm 2 yields a solvable inequalities system as shown below. We use the symbols $a$, $b$, $c$, and $e$ for the classes $A$, $B$, $C$, $D$ and $E$ respectively, $isa_1$, $isa_2$ $isa_3$ for the new associations between $A$ to $B$, $A$ to $C$ and $A$ to $D$ respectively, $isa_4$, $isa_5$ for the new associations between $E$ to $C$ and $E$ to $D$ respectively and the symbol $r$ for the $R$ association.

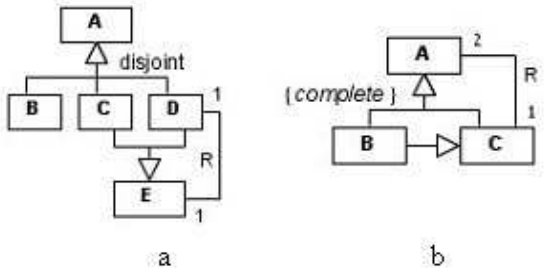


**Fig. 5.** Constrained Graph Hierarchy

1. **The Inequalities System produced by Algorithm 2 for Figure 5-a**:
   (a) **The Generalization Set** $\{A,\ B,\ C,\ D\}$:
   - $isa_1 = b,\ isa_1 \leq a$, $isa_2 = c,\ isa_2 \leq a$, $isa_3 = d,\ isa_3 \leq a$.
   - The disjoint inequality: $a \geq b + c + d$.

   (b) **The Generalization Set** $\{C,\ D,\ E\}$:
   - $isa_4 = c,\ isa_4 \leq e$, $isa_5 = d,\ isa_5 \leq e$, $r = d,\ r = e$.
2. **A Possible Solution**: $a = 3,\ b = c = d = e = 1,\ , isa_1 = isa_2 = isa_3 = isa_4 = isa_5 = 1,\ and\ r = 1$.

The reason for the failure of Algorithm 2 to detect that the diagram in Figure 5-a is not fully finitely satisfiable is lies in the projection of the *disjoint* constraint from one generalization set to the other. The implied *disjoint* constraint on the lower generalization set is not recorded in the inequalities system.

**Presence of a *complete* GS-Constraint in a [G-C]-GS class diagram**: Consider the class diagram in Figure 5-b. The *complete* constraint states that the union of the extensions of classes $B$ and $C$ is the extension of class $A$. Yet, $B$ is a subclass of $C$, implying that the extensions of $C$ and $A$ are equal. On other hand, the elements of class $C$ are mapped in a $1:2$ manner to those of class $A$.

The only solution for having a $1:2$, *onto* mapping from a set to itself is either empty or infinite. Therefore, the class diagram is not fully finitely satisfiable.

Nevertheless, Algorithm 2 yields a solvable inequalities system as shown below.

1. **The Inequalities System of Figure 5-b**:
   (a) $isa_1 = b,\ isa_1 \leq a, isa_2 = c,\ isa_2 \leq a, isa_3 = b,\ isa_3 \leq c, r = 2c,\ r = a.$
   (b) The completeness inequality: $a \leq b + c.$
2. **A Possible Solution**: $a = 2,\ b = c = 1,\ , isa_1 = isa_2 = isa_3 = 1,\ and\ r = 2.$

The reason for the failure of Algorithm 2 to detect that the diagram in Figure 5-b is not fully finitely satisfiable lies in the projection of the $\{B,\ C\}$ generalization set on the constraints imposed on the other generalization set. The implied constraint for the $\{A,\ B,\ C\}$ generalization set is *complete, overlapping*. The addition of this constraint yields an unsolvable inequalities system.

## 5 Conclusions and Future Work

In this paper, we have introduced a simple and effective method for deciding full finite satisfiability of class diagrams with constrained generalization sets. The advantage of this method lies in its simplicity and efficiency. The method applies to class diagram features that are not handled by other approaches, and improves the efficiency of existing methods.

We have studied the limits of this method with respect to the interaction between class hierarchy structure and the kind of *GS* constraints it includes. Yet, it seems that the combination of graph structured class hierarchies with the *disjoint* and *complete GS*-constraints does not occur that often. One possibility might be a combination of the expensive Calvanese-Lenzerini algorithm with our method. That is, apply our method in most cases, and resort to the inefficient method whenever our method does not apply. It is possible also that proper preprocessing of the *GS*-constraints in a class diagram, can strengthen our method.

In the future, we plan to explore the possible extension of the presented method for testing full finite satisfiability in the presence of n-ary association with complex cardinality constraints, qualifier constraints, association class constraints, and association constraints.

Another direction involves the possibility of expanding our method with heuristics for detecting and repairing finite satisfiability problems following the ideas of [10, 11]. The intention is to apply similar strategies for repairing finite satisfiability problems in UML2 class diagrams with class hierarchy constraints.

## 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] Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., and Rosati, R. . Description Logic Framework for Information Integration. In Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR'98), 2-13, (1998).
[7] Guizzardi, G., Wagner G., Guarino, N., van Sinderen, M.: An Ontologically well-Founded Profile for UML Conceptual Models. 16th International Conference on Advanced Information Systems Engineering (CAiSE), Latvia, (2004)
[8] Hartman, S.: Graph Theoretic Methods to Construct Entity-Relationship Databases. LNCS, Vol. 1017, (1995)
[9] Hartman, S.: On the Implication Problem for Cardinality Constraints and Functional Dependencies. Ann.Math.Artificial Intelligence, (2001)
[10] Hartman, S.: Coping with Inconsistent Constraint Specifications. LNCS, Vol. 2224, (2001)
[11] Hartman, S.: Soft Constraints and Heuristic Constraint Correction in Entity- Relationship Modeling. LNCS, Vol. 2582, (2002)
[12] Kaneiwa, K., Satoh, S.: Consistency Checking Algorithms for Restricted UML Class Diagrams. In Proceedings of the Fourth International Symposium on Foundations of Information and Knowledge Systems (2006)
[13] Kozlenkov, A., Zisman, A.: Discovering Recording, and Handling Inconsistencies in Software Specifications. Int. J. of Computer and Information Science 5(2), (2004)
[14] Lange, C., Chaudron, M., Muskens. J.: In Practice: UML Software Architecture and Design Description. IEEE Software, vol. 23, no. 2, (2006)
[15] Lenzerini, M. Nobili, P.: on the Satisfiability of Dependency Constraints in Entity-Relationship Schemata. Information Systems, Vol. 15, 4, (1990)
[16] Liang. P.: Formalization of Static and Dynamic UML Using Algebraic. Master's thesis, University of Brussel (2001)
[17] Maraee, A.: Efficient Methods for Solving Finite Satisfiability Problems in UML class Diagrams. Master' thesis, Ben-Gurion University of the Negev (2007)
[18] OMG.: UML 2.0 Superstructure Specification, (2005)
[19] Rumbaugh., J., Jacobson, G., Booch, G.: The Unified Modeling Language Reference Manual Second Edition. Adison Wesley (2004)
[20] Thalheim, B.: Entity Relationship Modeling, Foundation of Database Technology. Springer-Verlag, (2000)
[21] Unhelkar, B.: Verification and Validation for Quality of UML 2.0 Models. Addison-Wesley,(2005)