

# Efficient Recognition of Finite Satisfiability in UML Class Diagrams: Strengthening by Propagation of *Disjoint* Constraints \*

Azzam Maraee  
Ben-Gurion University of the Negev  
Beer-Sheva 84105, ISRAEL  
mari@cs.bgu.ac.il

Mira Balaban  
Ben-Gurion University of the Negev  
Beer-Sheva 84105, ISRAEL  
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 reasoning methods for verifying model correctness. This paper extends and strengthens our previous work on efficient recognition of finite satisfiability problems in UML class diagrams with constrained generalization sets. First, algorithm *FiniteSat* is simplified into a single stage process, yielding a more compact linear inequality system. The main contribution of the paper is a method for propagation of disjoint constraints within complex class hierarchy structures, which meaningfully extends the scope of the *FiniteSat* algorithm. The method relies on a thorough analysis of the interaction between disjoint constraints and the structure of class hierarchy. It is recommended as a pre-processing stage, and being an anytime algorithm, even partial application is useful.

## 1 Introduction

Unified Modeling Language (UML) is nowadays the widely accepted standard for system development. 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 systems structure in terms of classes, associations, and constraints imposed on classes and their inter-relationships. A class diagram is *consistent* if it can describe a non-empty reality (an *instance*) that satisfies all constraints; it is *finitely satisfiable* if that instance is finite. A consistent and finitely satisfiable class diagram is *correct* [6, 10, 16, 15].

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

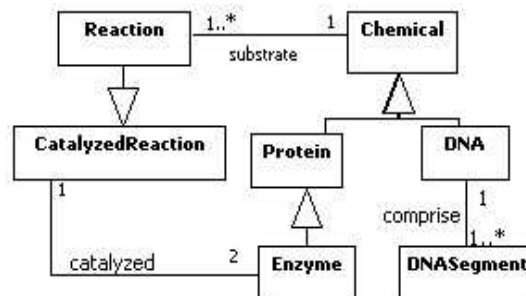


Figure 1. A Class Diagram with a Finite Satisfiability Problem

Class diagrams are models written by people, and therefore, they cannot guarantee correctness [3, 4, 8, 10, 16]. Problems arise in presence of constraints, which may turn the model incorrect, or may not be satisfied. Figure 1, presents a small ontology in 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  $C \geq 2C$ . Therefore, in every instance, *Chemical* denotes either an empty or an infinite set, implying that the class diagram is not *finitely satisfiable*.

Models lie at the heart of the emerging Model-driven Engineering approach, in which software is developed by repeated transformations of models [13]. 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 correct models, and to detect problems as early as possible. However, current CASE tools do not support reasoning about UML models, and enable the construction of erroneous ones. There is an urgent need for efficient methods for recognition, detection and repair of correctness problems in software models [4, 8, 13, 9, 7].

In our previous work [13], we presented an efficient, two stage algorithm, for deciding (recognizing) finite satisfiability in class diagrams with constrained generalization sets. In this paper we strengthen and extend our previous results. First, algorithm *FiniteSat* is simplified into a single stage process, yielding a more compact linear inequality system. The main contribution of the paper is a method for propagation of *disjoint* constraints within complex class hierarchy structures, which meaningfully extends the scope of the *FiniteSat* algorithm. The method relies on a thorough analysis of the interaction between disjoint constraints and the structure of class hierarchy. It is recommended as a pre-processing stage, and being an anytime algorithm, even partial application is useful.

Section 2 provides the relevant background. Section 3 presents the simplified finite-satisfiability decision procedure, and shortly proves its correctness. Sections 4 and 5 introduce the propagation of disjoint constraints method, and Section 6 concludes the paper.

## 2 Background

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

1. *Cardinality (multiplicity) 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. GSs are visualized as in Figure 2. GS constraints can be combined as follows:  $\{complete, disjoint\}$ ,  $\{incomplete,$

*disjoint\},  $\{complete, overlapping\}$ ,  $\{incomplete, overlapping\}$ .*

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  $CD$ .

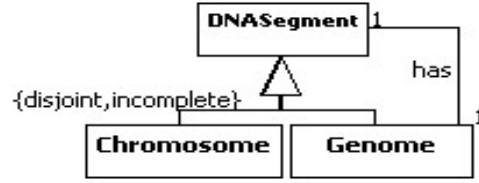


Figure 2. Constrained Generalization Set

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  $i, j$ , it might be  $C_i \cap C_j \neq \emptyset$ .

A *legal instance* of a class diagram is an instance that satisfies all constraints. Correctness of a class diagram involves *consistency* and *finite-at-satisfiability* [6, 10, 16, 3, 13]. 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 satisfiable<sup>1</sup>. 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 [12]. Class diagrams  $CD, CD'$  are *equivalent*, denoted  $CD \equiv CD'$ , if they have the same legal instances.

**Complexity:** Berardi et al., in [3], showed that deciding consistency of UML class diagrams is EXPTIME-complete. Artale et al. [1] refine these results, by considering fragments of class diagrams. They show that for ER diagrams

<sup>1</sup>Lenzerini and Nobili [10] used the notion of *strong satisfiability* for this term.

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. As for finite satisfiability, based on results of Lutz et al. [11], and of Schild [14], it can be concluded that deciding 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 infinity in unrestricted class diagrams is still an open issue. Below we shortly summarize results in the first approach, on which the *FiniteSat* algorithm is based.

The fundamental work in the linear inequalities approach is that of [10, 16]. It applies to *Entity-Relationship (ER)* diagrams with *Entity Types (Classes)*, *Binary Relationships*<sup>2</sup> (*Associations*), and *Cardinality Constraints*. The method transforms the cardinality constraints into a system of linear inequalities whose size is polynomial in the size of the diagram:

1. For a relationship  $r(rn_1 : C_1[\min_1, \max_1], rn_2 : C_2[\min_2, \max_2])$ , insert the inequalities:
  - For  $\min_2 > 0$ :  $r \geq \min_2 \cdot c_1$  and for  $\max_2 \neq * : r \leq \max_2 \cdot c_1$ .
  - 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:  $T > 0$ .

Calvanese and Lenzerini, in [6], extend the inequalities based method of [10] 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 [5] reduces the overall number of new class and association variables, but the worst case is still exponential.

<sup>2</sup>They allow also n-ary relationships, but with non-standard (membership) semantics for cardinality constraints.

## 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 [13] algorithm, which applies to class diagrams with *multiplicity constraints on binary associations*, *class hierarchy constraints*, and *Generalization Set (GS) constraints*.

The *FiniteSat* algorithm extends the algorithm of Lenzerini and Nobili [10] 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  $\Psi^{CD}$

**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  $GS(C, C_1, \dots, C_n; Const)$ ,  $C$  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 3-a, the directed class hierarchy is not a tree, as *F* is a subclass 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 3-b, class *F* has two *ISA* paths to its superclass *A*. The *ISA* path *A, B, F, C, A* forms an undirected *ISA* cycle.

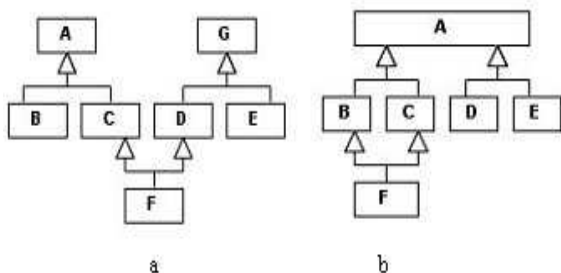


Figure 3. Unconstrained Hierarchy Structures

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 [10] 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 4 shows the reduced class diagram of Figure 1.

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

**Proof 1** (Sketched) *The reduction is defined (1) by bidirectional translations between non-empty finite legal instances I and I' of CD and CD', respectively. (2) By handling of class hierarchy [12, 2].*

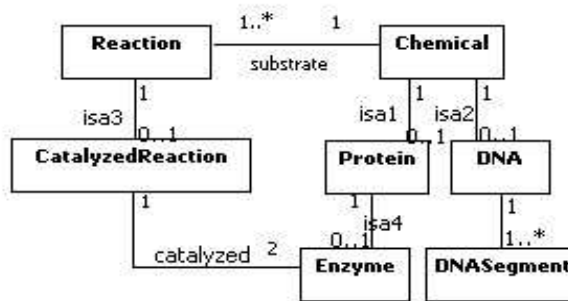


Figure 4. The Reduced Class Diagram of Figure 1

**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 2** (Sketched) *Given a class diagram CD, construct a class diagram CD' as above, to which the inequalities method of [10] is applied. Based on Lemma 1, CD is finitely satisfiable if and only if the inequality system of [10] for CD' is solvable. It is not hard to show that this inequality system is equivalent to the inequality system constructed by FiniteSat.*

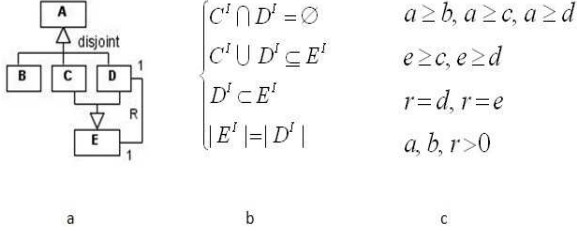
The results of this claim can be extended for class diagrams with GS constraints and acyclic class hierarchy structure, or cyclic structure in which class hierarchy cycles do not include *disjoint* or *complete* constraints. The scope of the *FiniteSat* algorithm is defined in the following claim:

**Claim 2** *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 3** *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.*

**Example 1** *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 as shown in Figure 5-b. The only solution for proper set inclusion with equal size is that the sets are either empty or infinite. Therefore, the diagram is not finitely satisfiable. Nevertheless, the **FiniteSat** Algorithm yields a solvable inequalities system as shown in Figure 5-c. The reason for the failure of **FiniteSat** lies in the projection of the disjoint constraint from the  $A$  GS to the  $E$  GS, which is not recorded in the inequality system.



**Figure 5. Finite Satisfiability Problem that is not Recognized by the FiniteSat Algorithm**

**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 4** Every constraint contributes a constant number of inequalities.

#### 4 Propagation of the *disjoint* GS constraint

In the previous section, it is shown that Algorithm **FiniteSat** fails to recognize non finite satisfiability for class hierarchies that include cycles with *disjoint* or *complete* constraints. In this section we present a method for propagation of *disjoint* constraints over cycles in the undirected class hierarchy graph. We show that the propagation extends the scope of the **FiniteSat** algorithm, and suggest to use it as a preprocessing stage.

##### Terminology:

The proper (strong) version of the class hierarchy relation  $\preceq$  (also termed *descendant* or *ISA*) is denoted  $\prec$ ; its transitive reflexive closure is denoted  $\preceq^*$ , and the transitive closure is  $\prec^+$ . These relations are extended to sets of classes, i.e.,  $S_1 \preceq S_2$  means that for every class  $A_1$  in  $S_1$  there is a class  $A_2$  in  $S_2$  such that  $A_1 \preceq A_2$ . A Class  $A$  is a *descendant* of a GS  $G$ , denoted  $A \preceq^* G$ , if  $A \preceq B$  for some subclass in  $G$ . The *descendant set* of a class  $A$ , denoted  $A^{\preceq^*}$ , is the set of classes  $A'$  such that  $A' \preceq A$ ;  $A^{\prec^+}$  is the set of proper descendants.  $A^{\preceq^* \text{-graph}}$  is the *descendant graph* of  $A$ , i.e., a directed graph whose nodes are the classes in  $A^{\preceq^*}$ , and its

edges are directed from a super-class to its direct descendant classes.

Two classes are *disjoint* if their extensions are disjoint in every legal instance of the class diagram. A set  $S$  of proper descendant classes of a class  $A$  that are pairwise disjoint, is called a *disjoint descendant set* of  $A$ , and denoted  $S \prec^{+dis} A$  (Note that every proper descendant of  $A$  is vacuously a disjoint descendant set of  $A$ ). The set of all disjoint descendant sets of a class  $A$ , denoted  $A^{\prec^{+dis}}$ , is partially ordered by the descendant relation  $\preceq$ . A generalization set  $G$  is a *logical implication* of a class diagram  $CD$ , denoted  $CD \models G$ , if  $I \models G$  in every legal instance  $I$  of  $CD$ .

**Proposition 1** If  $\{D_1, ..D_n\} \prec^{+dis} A$ , then  $CD \models GS(A, D_1, ..D_n; \{disjoint\})$ .

The following claim and example show that extending a class diagram with all logically implied disjoint GSs strengthens the **FiniteSat** algorithm. That is, **FiniteSat** is still sound, but can detect finite satisfiability problems that are not recognized in the original class diagram.

**Claim 4 [Full propagation of disjointness]** Let  $CD$  be a class diagram, and let  $\overline{CD}$  be  $CD$ , augmented with all GSs  $GS(A, D_1, ..D_n; \{disjoint\})$ , where  $\{D_1, ..D_n\}$  is in  $A^{\prec^{+dis}}$ , for all classes  $A$  in  $CD$ . Then: (1)  $CD \equiv \overline{CD}$ ; (2) Every solution of  $\Psi^{\overline{CD}}$  is a solution for  $\Psi^{CD}$ , but not vice versa.

**Proof 5** The class diagram equivalence results from Proposition 1. The inequality systems relation holds since  $\Psi^{\overline{CD}}$  includes  $\Psi^{CD}$ , and Example 2 shows that not all  $\Psi^{CD}$  solutions solve  $\Psi^{\overline{CD}}$  (implying that **FiniteSat** identifies more finite satisfiability problems when applied to  $\overline{CD}$ ).

**Example 2** Consider again Figure 5. Algorithm **FiniteSat** cannot detect its finite satisfiability problem. But, when the class diagram is augmented with  $GS(E, C, D; disjoint)$  (note that  $\{C, D\} \prec^{+dis} E$ ), **FiniteSat** adds the new inequality  $e \geq c + d$ , which accounts for the new disjoint constraint, and turns the whole inequality system unsolvable.

Consider Figure 6.a. By Claim 4, seven new disjoint GSs should be added:

$GS(A_1, A_2, A_4, A_5; disjoint)$ ,  $GS(A_1, A_2, A_4; disjoint)$ ,  $GS(A_1, A_2, A_5; disjoint)$ ,  $GS(A_1, A_8, A_4, A_5; disjoint)$ ,  $GS(A_1, A_4, A_5; disjoint)$ ,  $GS(A_3, A_4, A_5; disjoint)$ ,  $GS(A_7, A_8, A_4, A_5; disjoint)$ .

In general, in order to use the result of Claim 4, one has to augment a class diagram with GSs for all disjoint descendant sets of all classes. The problem is that there are too many such sets (exponential in the sizes of disjoint GSs and class hierarchy depths), and this extension is not feasible (and not reasonable). The following claim shows that a greatly restricted extension gives the same results.

**Claim 5 [Minimal propagation of disjointness]** Let  $CD$  be a class diagram, and let  $\overline{CD}$  be  $CD$ , augmented with all GSs  $GS(A, D_1, \dots, D_n; \{disjoint\})$ , where  $\{D_1, \dots, D_n\}$  is a maximal (top) element in  $A^{\prec^{+dis}}$ , for all classes  $A$  in  $CD$ . Then: (1)  $CD \equiv \overline{CD}$ ; (2)  $\Psi^{\overline{CD}} \equiv \Psi^{CD}$  (i.e., they have the same solutions).

**Proof 6**  $\overline{CD}$  is well defined since  $\preceq^*$  is a partial order, and therefore  $A^{\prec^{+dis}}$  has maximal elements. The class diagram equivalence results from Proposition 1. The inequality system equivalence is proved by noting first that if a GS for a set  $S$  in  $A^{\prec^{+dis}}$  is not added to  $\overline{CD}$ , then there is a set  $S'$  in  $A^{\prec^{+dis}}$ ,  $S \preceq^* S'$ , that is added to  $\overline{CD}$ . Then it is shown that the inequality generated by **FiniteSat** for the  $S'$  disjoint GS of  $A$ , implies the inequality that would have been generated for the  $S$  disjoint GS of  $A$ .

Claim 5 provides a feasible clue for strengthening **FiniteSat**: Compute  $\overline{CD}$  and apply **FiniteSat**. It shows that the result would be as strong as if we have created the full  $\overline{CD}$  class diagram. In order to compute  $\overline{CD}$  we need to find the maximal elements of  $A^{\prec^{+dis}}$ , for all classes  $A$ . But here we encounter a difficulty, due to the semantic based nature of  $A^{\prec^{+dis}}$  definition: Disjointness of classes can be caused by a variety of constraints, and their interaction. Yet, we know to identify only disjoint relations that are caused by explicit declarations of disjoint constraints. Therefore, Algorithm **Disjoint Propagation** below, considers only disjoint descendant sets that result from explicit disjoint constraints, implying that it actually computes an approximation of  $\overline{CD}$ . It might be the case that  $\overline{CD}$  includes disjoint GSs that are not detected by **Disjoint Propagation**, or that some GS redundancy is introduced since a maximal disjoint descendant set is missed.

In order to compute maximal disjoint descendant sets of classes, we introduce the *disjoint graph* of  $CD$ , denoted  $CD^{dis\_graph}$ , which is an undirected graph whose nodes are the classes of  $CD$ , and its edges connect nodes of disjoint classes. For example, the disjoint graph of Figure 6-a is given in Figure 6-b. The restriction of  $CD^{dis\_graph}$  to the set of proper descendants of a class  $A$  is denoted  $A^{dis\_graph} = CD^{dis\_graph} / A^{\prec^+}$ .

**Proposition 2** Top (maximal) elements of  $A^{\prec^{+dis}}$  are maximal cliques in  $A^{dis\_graph}$ .

Based on this proposition, Algorithm **Disjoint Propagation** computes top maximal cliques in the disjoint graphs of the descendants of classes. This proposition also accounts for its correctness, as computing an approximation of  $\overline{CD}$ :

## Algorithm 2 Disjoint Propagation

**Input:** A class diagram  $CD$ .

**Output:** A class diagram  $\overline{CD}$ , that possibly augments  $CD$  with additional disjoint GSs.

**Method:**

1. Create  $CD^{dis\_graph}$ .
2. For all classes  $A$  in  $CD$ :
  - (a) Create  $A^{dis\_graph}$ .
  - (b) Select the maximal cliques in  $A^{dis\_graph}$ .
  - (c) Order the maximal cliques by  $\preceq$ .
  - (d) Select the top elements in the ordered set of maximal cliques.
3. Extend  $CD$  with disjoint GSs for all selected sets of disjoint descendants, for all classes.

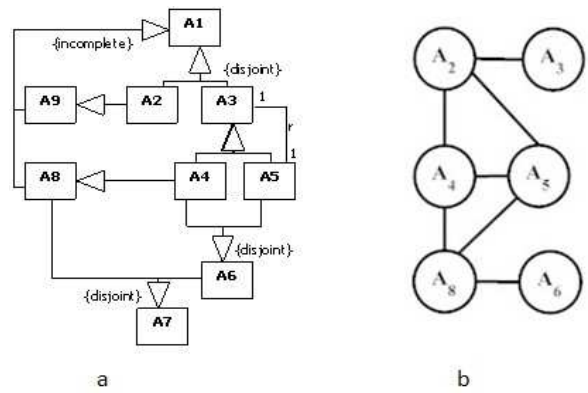


Figure 6. Propagation of the disjoint constraint

**Example 3** Applying **Disjoint Propagation** to the class diagram in Figure 6-a yields only two GSs instead of the seven GSs already shown. The steps are presented below.

**Step 1:** Creates the disjoint graph presented in Figure 6-b.

**Step 2:**

1. **Class  $A_1$ :**
  - **Step 2.a:**

$$A_1^{dis\_graph} = CD^{dis\_graph}_{\{A_2, A_3, A_4, A_5, A_8, A_9\}}$$
  - **Steps 2.b, 2.c, 2.d:** The ordered maximal cliques in  $A_1^{dis\_graph}$  are  $\{\{A_2, A_3\}, \{A_2, A_4, A_5\}, \{A_4, A_5, A_8\}\}$ . The top elements are  $\{\{A_2, A_3\}, \{A_4, A_5, A_8\}\}$ .
2. **Classes  $A_2, A_4, A_5$  have no descendants.**
3. **Class  $A_3$**

- Step 2.a:  $A_3^{dis\_graph} = CD_{\{A_4, A_5\}}^{dis\_graph}$
- Steps 2.b, 2.c, 2.d:  $A_3^{dis\_graph}$  includes a single clique  $\{A_4, A_5\}$ .

4. Class  $A_6$ : Same as class  $A_3$ .

5. Class  $A_7$ :

- Step 2.a:  $A_7^{dis\_graph} = CD_{\{A_4, A_5, A_6, A_8\}}^{dis\_graph}$ .
- Steps 2.b, 2.c, 2.d: The ordered maximal cliques in  $A_7^{dis\_graph}$  are  $\{\{A_6, A_8\}, \{A_4, A_5, A_8\}\}$ . The top element is  $\{A_6, A_8\}$ .

6. Classes  $A_8, A_9$ : Have a single descendant, each.

**Step 3:**

The algorithm extends  $CD$  with the GSs:  $GS(A_1, A_4, A_5, A_8; \{disjoint\})$  and  $GS(A_3, A_4, A_5; \{disjoint\})$ .

Consider again Figure 6-a. The extension of  $A_3$  properly includes the extension of  $A_5$ , and members of  $A_3$  are mapped in a 1:1 manner to members of  $A_4$ . Hence, the class diagram is not finitely satisfiable, but *FiniteSat* fails to detect that. Preprocessing *Disjoint Propagation* on this class diagram adds  $GS(A_3, A_4, A_5; disjoint)$ , which causes *FiniteSat* to create the inequality  $a_3 \geq a_4 + a_5$ , which turns the inequality system insolvable.

Assume the addition of two associations,  $r$  and  $q$ , between classes  $A_2$  and  $A_4$  to class  $A_1$ , respectively, as follows:

$$r(rn_1 : A_1[2, 2], rn_2 : A_2[1, 1]) \text{ and} \\ q(rn_1 : A_1[2, 2], rn_2 : A_4[1, 1]).$$

If  $a_1, a_2, a_4$  denote the size of  $A_1, A_2, A_4$ , respectively, then  $a_1$  must satisfy  $2 * a_1 = a_2$  and  $2 * a_1 = a_4$ , implying the inequality  $a_1 = a_2 + a_4$ . However, classes  $A_2$  and  $A_4$  are disjoint and descendants of the GS  $GS(A_1, A_8, A_9; incomplete)$ , implying that the extension of  $A_1$  properly includes the extensions of  $A_2$  and  $A_4$ . Therefore, these sets are either empty or finite. However, The inequality system after the *Disjoint Propagation* preprocessing is still solvable. The reason is that the *incomplete* constraint has a global effect in a cycle of class hierarchy that includes *disjoint* constraints, and *Disjoint Propagation* does not account for it. In the next section we strengthen *Disjoint Propagation* with propagation of  $\{disjoint, incomplete\}$  constraints, which is required in this case.

## 5 Propagation of The *incomplete, disjoint* Constraints

In this section we extend the previous propagation method to account for the interaction between *disjoint* and

*incomplete* constraints. A set of classes  $S$  is an *incomplete disjoint descendant set* of a class  $A$ , denoted  $S \prec^{+inc, dis} A$ , if  $S \prec^{+dis} A$  and in every legal instance  $I$ ,  $S^I \subset A^I$ . The set of all incomplete disjoint descendant sets of a class  $A$ , denoted  $A^{\prec^{+inc, dis}}$ , is partially ordered by the descendant relation  $\preceq$ .

### Proposition 3

If  $\{D_1, \dots, D_n\} \prec^{+inc, dis} A$ , then  $CD \models GS(A, D_1, \dots, D_n; \{incomplete, disjoint\})$ .

Similarly to the disjoint propagation, we first show that extending a class diagram with all logically implied incomplete disjoint GSs indeed strengthens the pre processing step of Algorithm *Disjoint Propagation* (Claim 6). That is, *FiniteSat* can detect finite satisfiability problems that are not detected following the pre-processing of *Disjoint Propagation*. Then we show that these results can be obtained with only minimal propagation of incomplete disjoint constraints (Claim 7). Finally, we extend Algorithm *Disjoint Propagation* with propagation of incomplete disjoint constraints (Algorithm 3). The incomplete disjoint propagation approximates the semantically defined minimal propagation, by considering explicitly defined incomplete constraints alone.

### Claim 6 [Full propagation of incomplete-disjoint and of disjoint]

Let  $CD$  be a class diagram, and let  $\overline{CD}^{inc}$  be  $CD$ , augmented with all GSs: (1)  $GS(A, D_1, \dots, D_n; \{incomplete, disjoint\})$ , where  $\{D_1, \dots, D_n\}$  is in  $A^{\prec^{+inc, dis}}$ ; (2)  $GS(A, D_1, \dots, D_n; \{disjoint\})$ , for all other  $\{D_1, \dots, D_n\}$  in  $A^{\prec^{+dis}}$ , for all classes  $A$  in  $CD$ . Then: (1)  $\overline{CD} \equiv \overline{CD}^{inc}$ ; (2) Every solution of  $\Psi^{\overline{CD}^{inc}}$  is a solution for  $\Psi^{\overline{CD}}$ , but not vice versa.

### Claim 7 [Minimal propagation of incomplete-disjoint and of disjoint]

Let  $CD$  be a class diagram, and let  $\overline{CD}^{inc}$  be  $CD$ , augmented with all GSs: (1)  $GS(A, D_1, \dots, D_n; \{incomplete, disjoint\})$ , where  $\{D_1, \dots, D_n\}$  is a maximal (top) element in  $A^{\prec^{+inc, dis}}$ ; (2)  $GS(A, D_1, \dots, D_n; \{disjoint\})$ , where  $\{D_1, \dots, D_n\}$  is a maximal (top) element in  $A^{\prec^{+dis}}$ , for all classes  $A$  in  $CD$ . Then: (1)  $CD \equiv \overline{CD}^{inc}$ ; (2)  $\Psi^{\overline{CD}^{inc}} \equiv \Psi^{\overline{CD}}$  (i.e., they have the same solutions).

Algorithm *Incomplete Disjoint Propagation* extends Algorithm *Disjoint Propagation* by adding *incomplete disjoint* GSs for cliques (in the disjoint graph) that are descendants of incomplete GSs in  $CD$ . The algorithm computes an approximation of  $\overline{CD}^{inc}$ , since membership in  $A^{\prec^{+inc, dis}}$  is determined by being descendants of incomplete GS constraints, thereby neglecting all other sources for essential incomplete covering of supersets.

### Algorithm 3 *Disjoint\_Incomplete Propagation*

**Input:** A class diagram  $CD$ .

**Output:** A class diagram  $\overline{CD}^{inc}$ , that possibly augments  $CD$  with additional disjoint or incomplete GSs.

**Method** (the extension steps are marked with \*):

1. Create  $CD^{dis\_graph}$ .
  2. For all classes  $A$  in  $CD$ :
    - (a) Create  $A^{dis\_graph}$ .
    - (b) (i) \* Choose, and order by  $\preceq$  cliques, whose members are descendants of incomplete GSs in  $CD$ .
    - (ii) \* Select the top elements in the ordered set of incomplete cliques.
  - (c) (i) Select the maximal cliques in  $A^{dis\_graph}$ .
  - (i) Order the maximal cliques by  $\preceq$ .
  - (ii) Select the top elements in the ordered set of maximal cliques.
3. For all classes, extend  $CD$  with disjoint GSs for all selected sets of disjoint descendants, and with incomplete disjoint GSs for all selected sets of incomplete disjoint descendants.

**Example 4** Applying **Incomplete Disjoint Propagation** to Figure 6-a yields the additional GS  $GS(A_1, A_2, A_4, \{disjoint, incomplete\})$ , which enables **FiniteSat** to recognize the finite satisfiability problem that is caused by the addition of the  $r$  and  $q$  association (added at the end of previous section). Since,  $A_1$  is the only class with non-empty incomplete disjoint descendant sets, we list the steps for  $A_1$  alone (all other steps are as described in Example 3).

**Class  $A_1$ :** Step 2.b.(ii) selects the cliques that are descendants of incomplete GSs. The only such subset is  $\{A_2, A_4\}$ , which is also the top element, and therefore selected and added to  $CD$ , as an incomplete disjoint GS.

Applying **FiniteSat** to the resulting class diagram yields the additional inequality  $a_1 > a_2 + a_4$ . This inequality, together with the previously explained inequality  $a_1 = a_2 + a_4$ , turn the inequality system insolvable.

## 6 Conclusion

In this paper, we described a meaningful extension to our previous work on efficient recognition of lack of finite satisfiability in class diagrams with constrained GSs. The extension is based on a preprocessing algorithm, that propagates *disjoint* and *incomplete, disjoint* constraints within cycles of class hierarchy constraints. Since the overall problem is known to be hard, there is a great value in extending the scope of efficient reasoning methods. Moreover, the

suggested preprocessing algorithms are anytime algorithms, meaning that even partial application is useful.

In the future, we plan to further extend the scope of class diagrams to which the **FiniteSat** algorithm applies. Still another planned extension is in the direction of *cause detection* and *repair*. Finally, our intention is to use this method for testing finite satisfiability of large class diagrams like the UML meta-model or the Java library classes.

## References

- [1] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyashev. Complexity of reasoning in entity relationship models. In *Proc. of the 2007 Description Logic Workshop (DL 2007)*, 2007.
- [2] M. Balaban and A. Maraee. A uml-based method for deciding finite satisfiability in description logics. In *Description Logics*, 2008.
- [3] D. Berardi, D. Calvanese, and D. Giacomo. Reasoning on uml class diagrams. *Artificial Intelligence*, 2003.
- [4] F. Boufares and H. Benaouer. Consistency problems in er-schemas for database systems. *Information Sciences*, 2004.
- [5] M. Cadoli, D. Calvanese, G. De Giacomo, and T. Mancini. Finite satisfiability of uml class diagrams by constraint programming. In *The Workshop on CSP Techniques with Immediate Application*, 2004.
- [6] D. Calvanese and M. Lenzerini. On the interaction between isa and cardinality constraints. In *The 10th IEEE Int. Conf. on Data Engineering*, 1994.
- [7] A. Egyed. Instant consistency checking for the uml. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 381–390. ACM, 2006.
- [8] S. Hartmann. Coping with inconsistent constraint specifications. In *Proceedings of the 20th International Conference on Conceptual Modeling*, pages 241–255, London, UK, 2001. Springer-Verlag.
- [9] K. Kaneiwa and S. Satoh. Consistency checking algorithms for restricted uml class diagrams. In *In Proceedings of the Fourth International Symposium on Foundations of Information and Knowledge Systems*, 2006.
- [10] M. Lenzerini and P. Nobili. on the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems*, 15(4), 1990.
- [11] C. Lutz, U. Sattler, and L. Tendera. The complexity of finite model reasoning in description logics. *Inf. Comput.*, 2005.
- [12] A. Maraee. Efficient methods for solving finite satisfiability problems in uml class diagrams. Master's thesis, Ben-Gurion University of the Negev, 2007.
- [13] A. Maraee and M. Balaban. Efficient reasoning about finite satisfiability of uml class diagrams with constrained generalization sets. In *The 3rd European Conference on Model-Driven Architecture*, 2007.
- [14] A. Schild. A correspondence theory for terminological logics: preliminary report. Technical Report KIT-BACK, FR 5-12, 1991.
- [15] M. Szlenk. Formal semantics and reasoning about uml class diagram. In *The International Conference on Dependability of Computer Systems*, 2006.
- [16] B. Thalheim. Fundamentals of cardinality constraints. In *The 11th International Conference on the Entity-Relationship Approach*, 1992.