Finite Satisfiability of UML Class Diagrams with Constrained Class Hierarchy

MIRA BALABAN, Ben-Gurion University of the Negev
AZZAM MARAEE, Ben-Gurion University of the Negev

Models lie at the heart of the emerging Model-driven Engineering approach. In order to guarantee precise, consistent and correct models, there is a need for efficient powerful methods for verifying model correctness. Class Diagram is the central language within UML. Its correctness problems involve issues of contradiction – the *consistency* problem, and issues of finite instantiation – the *finite satisfiability* problem.

This paper analyzes the problem of finite satisfiability of class diagrams with class hierarchy constraints and generalization set constraints. The paper introduces the ***FiniteSat*** algorithm for efficient detection of finite satisfiability in such class diagrams, and analyzes its limitations in terms of complex hierarchy structures. ***FiniteSat*** is strengthened in two directions. First, an algorithm for identification of the cause for a finite satisfiability problem is introduced. Second, a method for propagation of generalization set constraints in a class diagram is introduced. The propagation method serves as a preprocessing step that improves ***FiniteSat*** performance, and helps developers in clarifying intended constraints. These algorithms are implemented in the ***FiniteSatUSE*** tool [BGU Modeling Group 2011b], as part of our ongoing effort for constructing a model level integrated development environment ([BGU Modeling Group 2010a]).

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*; *Object-oriented design methods*

General Terms: Design, Languages, Reliability, Verification

Additional Key Words and Phrases: Class hierarchy constraints, class hierarchy structure, detection and cause identification, finite satisfiability, generalization set constraints, identification graph, multiplicity constraints, solvability of linear inequality system, UML class diagram.

ACM Reference Format:
Balaban, M., Maraee, A. 2010. Finite Satisfiability of UML Class Diagrams with Constrained Class Hierarchy. ACM Trans. Softw. Eng. Methodol. V, N, Article A (January YYYY), 45 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Models lie at the heart of the emerging Model-driven Engineering approach, in which software is developed by repeated transformations of models [Kleppe et al. 2003]. In this paradigm, the software development process consists of models that are combined, refined, translated and integrated, to produce a final software product. That is, models are no longer restricted design artifacts, but play a central role in the development life

Supported in part by the Paul Ivanir Center for Robotics and Production Management at Ben- Gurion University of the Negev.
This article is a revised and extended version of papers presented in ECMDA 2007 and MBSE 2009.
Author's addresses: M. Balaban and A. Maraee, Computer Science Department, Ben-Gurion University of the Negev; email: {mira,mari}@cs.bgu.ac.il.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

cycle of software. Therefore, it is essential to have precise, consistent and correct models. Models should provide reliable support for the designed systems, and be subject to stringent quality verification and control criteria. Yet, current CASE (Computer Aided Software Engineering) tools enable the construction of erroneous models [Chanda et al. 2010]. Hence, there is an urgent need for efficient methods for verifying model correctness.

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.

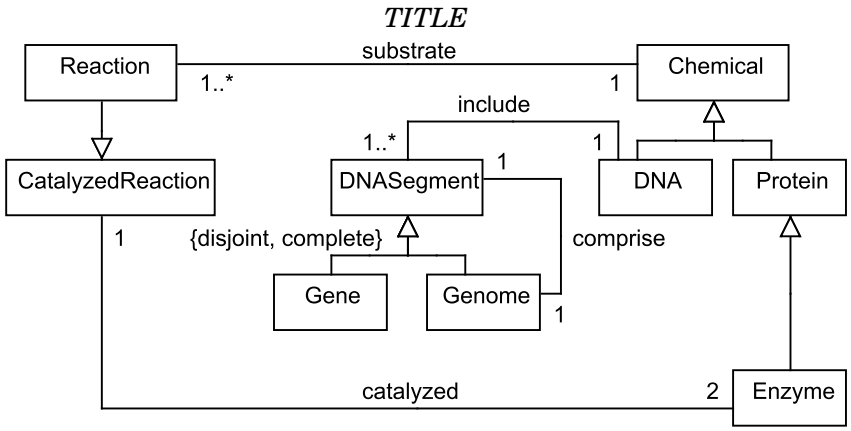


Fig. 1: A Class Diagram with a Finite Satisfiability Problem [1]

Class diagrams are written by people, and therefore cannot guarantee correctness and consistency [Sunye et al. 2001; Lange et al. 2006]. Problems usually arise in presence of constraints, which may turn the model inconsistent. 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.

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 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 instance[2], but in every such instance, all classes in the mentioned above cycle denote infinite sets, implying that the class diagram is not *finitely satisfiable*.

[1]Based on a class diagram that appears in `www.ccs.neu.edu/home/kenb/pub/2002/03/public.ppt`.

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

The problem of finite satisfiability frequently arises in large class diagrams that include non-trivial multiplicity constraints (different than 0, 1, *). Such class diagrams are used, for example, for modeling configuration management systems [European Rail Traffic Management System 2007], Biological systems, data-bases [Thalheim 2000; Blaha and Premerlani 1997; Spaccapietra 1987] and meta-modeling. In the area of configuration management (e.g., railway interlocking systems and network monitoring), system structure is modeled by large class diagrams that involve complex constraints, including non-trivial multiplicity constraints, class hierarchy with multiple inheritance constraints, generalization-set constraints with disjoint and complete restrictions, and association-class, xor, subsetting, redefinition and union constraints[3] [Fleishanderl et al. 1998; Felfernig et al. 2001; Georg et al. 2001; Bayley 2004; Falkner et al. 2010; Feinerer et al. 2011]. Similar complex constraints appear in *Extended-Entity-Relationship* (*EER*) and class diagram models of data bases, in Genome modeling systems [Paton et al. 2000] and in the UML meta-model [OMG 2007]. Complex constraints as listed above occur also in translations of description logic knowledge bases to UML class diagrams [Berardi et al. 2005; Balaban and Maraee 2008]. [Makarenkov et al. 2009] shows the high frequency of occurrence of the finite satisfiability problem in large synthetic class diagrams with non-trivial multiplicity constraints.

The primary method for detecting existence of finite satisfiability problems in class diagrams reduces the finite satisfiability problem to the problem of solvability of a system of linear inequalities. The main approach for identification of the cause for finite satisfiability problems involves identification of infinity causing association cycles in the diagram.

Fundamental work on this problem is that of [Lenzerini and Nobili 1990; Thalheim 1992]. It applies to *Entity-Relationship Diagrams* (*ERD*s) with *entity types* (*classes*), *binary relationships*[4] (*associations*), and *multiplicity constraints* [Chen 1976; Thalheim 2000]. The complexity is polynomial in the size of the diagram, both for detecting a finite satisfiability problem and for identifying its causes. Calvanese and Lenzerini, in [Calvanese and Lenzerini 1994], extend the inequalities based method of [Lenzerini and Nobili 1990] to apply to diagrams with class hierarchy constraints, but the size of the resulting system of inequalities is exponential in the size of the class diagram.

In this paper we introduce efficient methods for reasoning about finite satisfiability of class diagrams with class hierarchy constraints and generalization set constraints. First, we extend the linear inequality method of [Lenzerini and Nobili 1990], to apply to such constraints. This is the ***FiniteSat*** algorithm. The extension depends on the *structure of class hierarchy* constraints in the diagram, which is a new novel parameter, not considered earlier elsewhere. Two structures are distinguished: *non-cyclic class hierarchies* and *cyclic class hierarchies*, which are possible only in the presence of multiple inheritance. The ***FiniteSat*** algorithm is *sound* and *complete* if class hierarchy cycles do not include *disjoint* or *complete* constraints. Otherwise, ***FiniteSat*** is sound but incomplete.

In order to improve the results for such structures, ***FiniteSat*** is strengthened with a semantics preserving propagation of *disjoint* and of *incomplete* constraints, as a preprocessing transformation that is applied to the class diagram prior to the application of ***FiniteSat***. The propagation algorithms can also provide help to developers by highlighting implicit constraints.

FiniteSat shows that for UML class diagrams with multiplicity constraints, class hierarchies and generalization set constraints, but without class hierarchy cycles that

[3]See [OMG 2007] for specification of these constraints.

[4]They allow also n-ary relationships, but with non-standard (membership) semantics for cardinality constraints [Balaban and Shoval 2002].

include *disjoint* or *complete* constraints, finite satisfiability can be decided in polynomial time. In any case, the efficient ***FiniteSat*** algorithm cannot apply to all UML class diagrams since deciding finite satisfiability is EXPTIME-complete. For such cases, we suggest considerings relaxation of class hierarchy constraints.

Detection of finite satisfiability is strengthened by identification of the cause for a finite satisfiability problem. The method of Lenzerini and Nobili is extended to apply to class hierarchy and generalization set constraints. These methods are implemented within the ***FiniteSatUSE*** tool [BGU Modeling Group 2011b]. Examples of application in various domains, including performance evaluation, appear in [BGU Modeling Group 2011a]. Interestingly, the tool was able to detect and identify the cause for finite satisfiability problems that were not detected by expert modelers. These methods are integrated into our ongoing effort for constructing a catalog of design patterns and anti-patterns for class diagrams [Balaban et al. 2010; BGU Modeling Group 2010b], in order to produce a human understandable explanation and a repair advice.

Section 2 describes UML class diagrams and provides background on reasoning about their finite satisfiability. Section 3 introduces the ***FiniteSat*** algorithm and proves its correctness and scope. Section 4 introduces the algorithm for propagation of *disjoint* and *incomplete* constraints. Section 5 describes the cause identification method, and Section 6 concludes the paper and discusses future directions.

2. BACKGROUND

A class diagram is a structural abstraction of a real world phenomenon. The model consists of basic elements, descriptors and constraints. The basic elements are classes and associations, the descriptors are class and association attributes, and the constraints are restrictions imposed on these elements. The constraints are (1) multiplicity (cardinality) constraints on associations, with or without qualifiers; (2) association class constraint; (3) class hierarchy constraints; (4) generalization set constraints; (5) inter association constraints; (6) aggregation constraints; and (7) multiplicity (cardinality) constraints on attributes.

Figure 1 is an example of a class diagram, which partially specifies an ontology in the molecular biology domain. It consists of classes *Chemical, Reaction, CatalyzedReaction, Enzyme, DNA, DNASegment, Protein*, and associations *substrate, catalyzed, comprise*. It includes *multiplicity*, *class hierarchy* (*generalization, specialization*), and *Generalization set* constraints. Instances of *Chemical* must be related to *Reaction* instances, which are also instances of *CatalyzedReaction*, whose instances must be related to *Enzyme* instances, which are also instances of *Protein* and of *Chemical*. Instances of *DNA* are instances of *Chemical* as well, and must be related to *DNASegment* instances. The classes *DNASegment, Genome, Chromosome* form a generalization set with a *disjoint, complete* constraint.

The visual notation of class diagrams is considered as their *concrete syntax*, i.e., their precise representation. It is abstracted by the *abstract syntax*, which formally defines the semantically meaningful syntactical categories, and their inter-relationships. The semantics is defined for the abstract syntax. Below, we define the abstract syntax and the semantics of class diagrams. For clarity, we demonstrate the abstract syntax using the visual concrete syntax.

Abstract syntax. The subset of UML2.0 class diagrams considered in this paper includes *classes* (possibly with attributes), *associations* and three kinds of constraints: *Multiplicity* constraints on binary associations, *Class hierarchy* constraints, and *Generalization set* (*GS*) constraints. The formulation is based on the meta-model notion of *Property*, which is more basic than that of *Association* [OMG 2007; Alanen and Porres

2008]. The semantics is set-theoretic. In the rest of this paper attributes are omitted as there are no attribute constraints.

A *class diagram* is a tuple $\langle \mathcal{C}, \mathcal{P}, \mathcal{A}, props, source, target, \mathcal{C}onstraint \rangle$ where

— $\mathcal{C}$ is a set of *class* symbols.
— $\mathcal{P}$ is a set of *property* symbols (sometimes called *association end*s). Property symbols denote mappings derived from their associations.
— $\mathcal{A}$ is a set of *association* symbols.
— $props\colon \mathcal{A} \to \mathcal{P} \times \mathcal{P}$ is a 1:1 and onto assignment of (unique) properties to association symbols. For a property p, there is a unique $a \in \mathcal{A}$, such that $props(a) = (p, *)$ or $props(a) = (*, p)$, where * is a wild card. We write $assoc(p)$ or $assoc(p_1, p_2)$ for the association of p or of (p_1, p_2), and $props_1(a), props_2(a)$ for the two properties of a.
— $source\colon \mathcal{P} \to \mathcal{C}$ and $target\colon \mathcal{P} \to \mathcal{C}$ are 1:1 mappings of properties to classes such that for an association a with $props(a) = (p_1, p_2)$, $target(p_1) = source(p_2)$ and $target(p_2) = source(p_1)$. In Figure 2, $target(p_1) = source(p_2) = C_1$ and $source(p_1) = target(p_2) = C_2$.

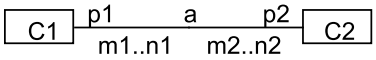


Fig. 2: Visualization of a binary association, its properties, their source and target classes, and their multiplicities

— $\mathcal{C}onstraint$ is a set of *constraints* as follows:
(1) *Multiplicity constraints on properties:* $mul\colon \mathcal{P} \to (\mathbb{N} \cup \{0\}) \times (\mathbb{N} \cup \{*\})$ assigns multiplicity constraints to property symbols. For simplicity we use a compact symbolic representation, where association a in Figure 2 is denoted $a(p_1 : C_1[\, m_1,\, n_1],\, p_2 : C_2[\, m_2,\, n_2])$. The functions $minMul\colon \mathcal{P} \to \{\mathbb{N} \cup \{0\}\}$ and $maxMul\colon \mathcal{P} \to \{\mathbb{N} \cup \{*\}\}$ give the minimum and maximum multiplicities assigned to a property, respectively.
(2) *Class-hierarchy:* A non-circular binary relationship $\prec$ on the set of class symbols: $\prec \ \subseteq \mathcal{C} \times \mathcal{C}$. Henceforth we use the notation $C_2 \prec C_1$, where C_1 is the superclass and C_2 is the subclass (also called *direct-descendant*).
The weak version of $\prec$ is denoted $\preceq$, which is "$\prec$ or equal". The reflexive transitive closure of $\prec$ is called the *descendant* relation, and denoted $\prec^*$. Its irreflexive version is denoted $\prec^+$. Figure 3a shows the concrete (visual) syntax of a class hierarchy constraint.
(3) *Generalization-set* (*GS*) *constraints:* GS is an $(n+1)$-ary relationship on $\mathcal{C}$, for $n \geq 2$. An element $\langle C, C_1, \ldots, C_n \rangle$ in GS must satisfy:
For $i, j = 1..n$ (1) $C \neq C_i$; (2) $C_i \neq C_j$; (3) $C_i \prec C$.
C is called the *superclass* and the C_i-s are called the *subclasses*. Elements of GS maybe associated with a *constraint* $const \in \{\langle disjoint \rangle, \langle overlapping \rangle, \langle complete \rangle, \langle incomplete \rangle, \langle disjoint, complete \rangle, \langle disjoint, incomplete \rangle, \langle overlapping, complete \rangle, \langle overlapping, incomplete \rangle\}$. We use the symbolic representation $GS(C, C_1, \ldots, C_n; const)$ for GS constraints. Note that an unconstrained *GS* is redundant, as it specifies only class hierarchy constraints. Figure 3b shows the concrete (visual) syntax of a GS constraint.

Henceforth we omit the term "symbol", and refer just to *classes, associations, properties*. 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.

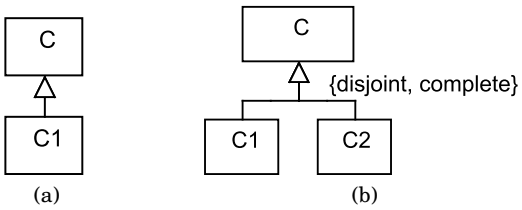


Fig. 3: Visualization of class-hierarchy and GS constraints

Semantics. The standard set theoretic semantics of class diagrams associates a class diagram with *instances* I, that have a semantic domain and an *extension mapping*, that associates syntactic symbols with elements over the semantic domain. For a symbol x, x^I is its denotation in I.
Symbol denotation:

— **Class:** For a class C, C^I, the extension of C in I, is a set of elements in the semantic domain. The elements of class extensions are called *objects*.
— **Property:** For a property p, p^I is a multi-valued function from its source class to its target class: $p^I : source(p)^I \rightarrow target(p)^I$.
— **Association:** For an association a, a^I, the extension of a in I, is a binary relationship on the extensions of the classes of a. If $props(a) = (p_1, p_2)$, then p_1^I and p_2^I are restricted to be inverse functions of each other. That is, $p_1^I = (p_2^I)^{-1}$. The association denotes all object pairs that are related by its properties. That is, $a^I = \{(e, e') \mid e \in target(p_1)^I,\ e' \in target(p_2)^I,\ p_2^I(e) = e'\}$. The elements of association extensions are called *links*.

The semantics of the constraints with respect to an instance I is defined as follows:

(1) **Multiplicity constraints on properties:** For a property p, p^I is restricted by the multiplicity constraints. For every $\mathtt{e} \in source(p)^I$, $minMul(p) \leq |p^I(\mathtt{e})| \leq maxMul(p)$. The upper bound constraint is ignored if $maxMul(p) = *$.
(2) **Class-hierarchy constraints:** Class hierarchy constraints denote subset relations between the extensions of the involved classes. That is, for $C_1 \prec C_2$, ${C_1}^I \subseteq {C_2}^I$.
(3) **GS constraints** have the following meaning:
   (a) *disjoint:* $C_i^I \cap C_j^I = \emptyset, \forall i \neq j$
   (b) *overlapping:* For some i, j, it might be $C_i^I \cap C_j^I \neq \emptyset$
   (c) *complete:* $C^I = \bigcup_{i=1}^{n} C_i^I$
   (d) *incomplete*: $\bigcup_{i=1}^{n} C_i^I \subseteq C^I$

   The meaning of combined constraints is the combined meaning of the individual constraints. According to the official semantics of the *incomplete* and *overlapping* constraints [OMG 2007] they are "non-constraints", in the sense that they do not impose any restriction: The subclasses might cover or not and might be overlapping or not, respectively. The semantics adopted by the ***FiniteSat*** algorithm introduced in Section 3, requires that these constraints are properly satisfied in at least one legal instance.

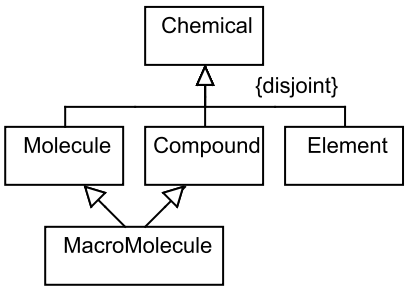


Fig. 4: An Inconsistent Class Diagram

A *legal instance* of a class diagram is an instance that satisfies all constraints. Class diagrams $CD,\ CD'$ are *equivalent*, denoted $CD \equiv CD'$, if they have the same legal instances.

2.1. Correctness of Class Diagrams

Correctness of a class diagram involves *consistency* [Andre et al. 2000; Berardi et al. 2005; Queralt and Teniente 2008; Artale et al. 2010; Kaneiwa and Satoh 2010; Jarrar and Heymans 2008] and *finite-satisfiability* [Lenzerini and Nobili 1990; Thalheim 2000; Calvanese and Lenzerini 1994; Boufares and Bennaceur 2004; Maraee and Balaban 2007; Cabot et al. 2008; Berrabah and Boufarès 2008].

Consistency. 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. Figure 4 presents an inconsistent class diagram. Inconsistency is caused by the contradiction between the *disjoint GS* constraint and the multiple inheritance constraint, which imply that the *MacroMolecule* class is empty in every legal instance.

Finite Satisfiability. Finite satisfiability means that a class has a finite non-empty extension. Figure 1 presents an example of a finite satisfiability problem. The problem is caused by interaction between multiplicity and class hierarchy constraints. Finite satisfiability is an essential property in software and in databases, since a necessarily empty or infinite class extension means that the class cannot be populated. 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[5]. Note that finite satisfiability implies consistency, but not vice versa.

THEOREM 2.1. *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.*

PROOF. (Sketched) This property results from the semantics of class diagram constraints: For all constraints, the union of disjoint legal instances of a class diagram yields a legal instance [Maraee 2007]. Note that this is not the case for general OCL constraints. □

Complexity. Berardi et al., in [Berardi et al. 2005], show that deciding consistency of a class in a UML class diagram is EXPTIME-complete. Artale et al. [Artale et al. 2007] refine these results, by considering fragments of class diagrams. They show that for ER diagrams that include multiplicity, class hierarchy and *disjoint* constraints, deciding consistency of a class is in NLogSpace. Addition of *complete* constraints raises

[5]Lenzerini and Nobili [Lenzerini and Nobili 1990] use the term *strong satisfiability*.

the complexity to NP, and addition of association hierarchy reaches the EXPTIME-complete complexity.

Consistency of class diagrams is studied in [Kaneiwa and Satoh 2010; Artale et al. 2010]. [Kaneiwa and Satoh 2010] provide upper bounds by introducing optimized algorithms for deciding class diagram consistency for various combinations of constraints. Their algorithms range from P (no *complete* constraints) to EXPTIME. [Artale et al. 2010] tightens these bounds to range between NLogSpace-complete to EXPTIME-complete.

In [Lutz et al. 2005] it is shown that finite satisfiability of the description logic *ALCQI* is EXPTIME-complete. Based on [Schild 1991], the same result holds for the description logic *ALC*. Since the reductions in [Berardi et al. 2005], from *ALC* to UML class diagrams, and from class diagrams to *ALCQI*, preserve finite satisfiability, finite satisfiability of class diagrams is also EXPTIME-complete.

2.2. Reasoning about Finite Satisfiability of Class Diagrams

Detection. The method of [Lenzerini and Nobili 1990; Thalheim 1992] transforms multiplicity constraints into a system of linear inequalities. It applies to *Entity-Relationship Diagrams* (*ERD*s) with *entity types* (*classes*), *relationships* (*associations*), and *multiplicity constraints on binary associations*. The inequalities are constructed according to the following rules:

(1) For every entity type (a class) E, insert a variable e and the inequality: $e > 0$.
(2) For every relationship type (an association) R, insert a variable r and the inequality: $r \geq 0$.
(3) For multiplicity constraints $r(rn_1 : C_1[min_1, max_1],\ rn_2 : C_2[min_2, max_2])$, 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 $max_1 \neq * :\ r \leq max_1 \cdot c_2$.

Their main result is that a diagram is finitely satisfiable iff the resulting system of inequalities is solvable. Therefore, this method can be used for *detecting* a problem of finite satisfiability of class diagrams with multiplicity constraints on binary associations. It has polynomial complexity since the size of the inequality system is polynomial in the size of the diagram, and solving a linear inequality system is polynomial in the encoding of the system size [Schrijver 1998].

The problem detection method of [Lenzerini and Nobili 1990] is extended, in [Calvanese and Lenzerini 1994], 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 associations accordingly. Therefore, the size of the resulting system of inequalities is exponential in the size of the class diagram. The simplification of [Cadoli et al. 2004] reduces the overall number of new class and association variables, but the worst case is still exponential.

Identification. A method for *identification* of the cause for a finite satisfiability problem is suggested in [Lenzerini and Nobili 1990]. The method is based on construction of an *identification graph*, 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 5. Cycles whose weights are smaller than 1 (the *weight of a path* is the product of the weights of its edges), are termed *critical*. A critical cycle identifies a set of multiplicity constraints that cause a finite satisfiability problem.

Similar approaches are introduced in [Thalheim 2000; Hartmann 1995, 2001]. [Engel and Hartman 1995] present graph based algorithms for detection and identification

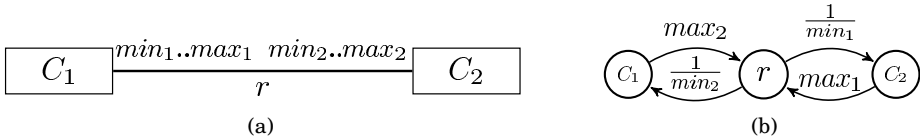


Fig. 5: The identification graph of a binary association in [Lenzerini and Nobili 1990]

of finite satisfiability problems in ER diagrams. The methods include an algorithm for construction of a *minimal legal instance* (an instance whose classes or associations cannot be decreased without violating a constraint). [Hartmann 2001] provides heuristic strategies for automatic repair of an identified finite satisfiability problem.

The above methods for detection and identification of finite satisfiability apply only to fragments of UML class diagrams. The methods of [Lenzerini and Nobili 1990] apply only to multiplicity constraints on binary associations, and [Calvanese and Lenzerini 1994] extends their detection method to apply to class hierarchy constraints. [Feinerer 2007; Falkner et al. 2010] extend the [Lenzerini and Nobili 1990] detection and identification methods to apply to class diagrams with *n-arry* associations and with binary associations that are annotated as *non-unique* (see [OMG 2007]). Similarly to [Engel and Hartman 1995], they also provide an algorithm for creating a minimal legal instance. They apply their methods to practical configuration management problems. Deciding finite satisfiability in unrestricted UML class diagrams is still an open issue.

[Formica 2002] presents a graph based method for detection of finite satisfiability in object-oriented database schemas. The method identifies finite satisfiability problems that result from interaction of multiplicity constraints on attributes, with other integrity constraints.

Reasoning about Class Diagrams with OCL Constraints. The expressiveness of class diagrams is limited to the set of constraints included in the syntax of the Class Diagram language. The *Object Constraint Language* (*OCL*) [Object Management Group 2006; Warmer and Kleppe 2003] extends UML models (mainly class diagrams) with general symbolic constraints. In particular, the class hierarchy and GS constraints can be expressed as OCL constraints[6]. OCL has the expressive power of first order logic, which turns analysis of UML/OCL undecidable. Therefore, most works employ bounded reasoning.

[Queralt and Teniente 2006, 2008] and AuRUS [Queralt et al. 2010] present reasoning over deductive databases that are derived from UML/OCL diagrams. The diagram can include multiple inheritance, and GS and association class constraints. They deal with problems of schema satisfiability, class satisfiability, constraint redundancy, and partial instance determination. AuRUS also identifies OCL constraints that enable terminated reasoning.

Alloy is a textual modeling language based on relational logic [Jackson 2002, 2006]. The Alloy analyzer [Jackson and Rinard 2004] employs SAT solving for bounded validation of user assertions. Together with the recent translation tools of UML/OCL class diagrams (UML2Alloy [Anastasakis et al. 2010], CD2Alloy [Maoz et al. 2011]) it provides a UML/OCL analysis tool that supports instance generation and completion. Nevertheless, the translations of UML class diagrams to Alloy do not support the full set of class diagram constraints (UML2Alloy does not support multiple inheritance or composition; CD2Alloy does support these features, but does not support

[6]Using OCL constraints that impose subset relations, set disjointness and use the set union operator. Note that replacing class hierarchy constraints by associations with multiplicity constraints, as suggested in [Gogolla and Richters 2002], does not preserve non-finite-satisfiability, as shown in the proof of Theorem 3.6

GS constraints and other features, such as association classes, and inter-association constraints).

USE is a plugin system that supports validation of UML/OCL models, and the verification tasks of proving consistency, independence of invariants, and checking consequences [Gogolla et al. 2005, 2009]. USE supports multiple inheritance, association classes and inter-association constraints, but does not support GS constraints. Recently, USE has been extended with a model validator plugin that employs off-the-shelf SAT solvers [Soeken et al. 2010; Kuhlmann et al. 2011].

HOL-OCL [Brucker and Wolff 2006, 2008] is a rich theorem proving environment for proving properties of UML diagrams that are annotated with OCL invariants. It supports user interactive proofs of consistency, instance validation and general consequence derivation.

UMLtoCSP [Cabot et al. 2007, 2008] is a CSP (Constraint Satisfaction Problem) based tool for reasoning about UML/OCL class diagrams. The tool supports bounded reasoning about satisfiability, consistency, finite satisfiability, independence of invariants and partial state completion. It handles class diagrams with multiplicity, class hierarchy, GS and association class constraints, but does not allow multiple inheritance[7]. The bounded reasoning mode does not enable general analysis. For example, it is possible to infer that a class diagram is not satisfiable within a given class size bound, but not that it is overall finitely satisfiable. [Shaikh et al. 2010, 2011] present slicing techniques for UML/OCL class diagrams, that ensure preservation of satisfiability and finite satisfiability. The procedures are implemented on top of the UMLtoCSP tool, and improve the scalability of the verification process. A further experiment (employing Alloy) shows that the slicing procedure is independent both of the tool and of the formalism. [Wahler et al. 2010] introduces an implemented set of OCL patterns that capture real business constraints.

CLEWS [Falkner et al. 2010; Feinerer et al. 2011; Niederbrucker, G. and Sisel, T. 2011] is a tool for reasoning about configuration management systems. It supports incremental identification of finite satisfiability problems in such systems (a single cause for an identified problem). It handles class diagrams with multiplicity and unique/non-unique constraints, that are augmented with domain specific constraints. In addition, the tool supports instance verification and completion, minimal instance creation and a form of reasoning about redundancy of multiplicity constraints, that is dedicated to configuration management.

To the best of our knowledge, the only publicly available tools that support finite satisfiability analysis are UMLtoCSP and CLEWS. The UMLtoCSP tool and our FiniteSatUSE tool are orthogonal in the sense that the first supports only bounded finite satisfiability analysis for UML/OCL class diagrams without multiple inheritance, while the latter supports full finite satisfiability analysis but for UML class diagrams without associated OCL invariants. The CLEWS and FiniteSatUSE tools seem to complement each other: Both tools provide full finite satisfiability analysis, but for different subsets of UML class diagrams without OCL constraints. CLEWS provides additional services that are dedicated to configuration problems. FiniteSatUSE is integrated within the USE system, while CLEWS is a standalone application.

3. EFFICIENT DETECTION OF FINITE SATISFIABILITY

In this section we describe the ***FiniteSat*** efficient algorithm (Algorithm 1) for deciding finite satisfiability in UML class diagrams and prove its correctness. The scope of the algorithm is defined by the structure of the class hierarchy in a knowledge base, rather

[7]A problem that appears both in UMLtoCSP and in USE, relates to hierarchy of association classes. It is explained in [Maraee and Balaban 2011].

than by a fragment of the language. Earlier partial versions appeared in [Maraee and Balaban 2007; Maraee et al. 2008].

ALGORITHM 1: *FiniteSat*

Input: A class diagram CD with binary multiplicity constraints, class hierarchy constraints, and GS constraints.
Output: A linear inequality system Ψ^{CD}.
begin
Insert a variable for every class and association in CD.
(1) For every multiplicity constraint, insert inequalities according to the Lenzerini and Nobili method (see Section 2).
(2) For every class hierarchy $B \prec A$ constraint, B being the subclass with variable b, and A being the superclass with variable a, add the inequality $b \leq a$.
(3) For every GS constraint $GS(C, C_1, ...C_n;\ Const)$, C being the superclass with variable c, C_is being the subclasses with variables c_i, and $Const$ being the GS constraint, add the following inequalities:
— ***Const*** = $\langle disjoint \rangle$: $c \geq \sum_{j=1}^{n} c_j$
— ***Const*** = $\langle overlapping \rangle$: Without inequality
— ***Const*** = $\langle complete \rangle$: $c \leq \sum_{j=1}^{n} c_j$
— ***Const*** = $\langle incomplete \rangle$: $\forall j \in [1, n].c > c_j$
— ***Const*** = $\langle disjoint, complete \rangle$: $c = \sum_{j=1}^{n} c_j$
— ***Const*** = $\langle disjoint, incomplete \rangle$: $c > \sum_{j=1}^{n} c_j$
— ***Const*** = $\langle overlapping, complete \rangle$: $c < \sum_{j=1}^{n} c_j$
— ***Const*** = $\langle overlapping, incomplete \rangle$: $\forall j \in [1, n].c > c_j$
Note that the GS constraint implies n class hierarchy constraints $C_i \prec C,\ i = 1..n$, for which n inequalities are inserted as in (2) above.
end

The inequalities introduced for the *incomplete* and *overlapping* constraints require the existence of a finite non-empty legal instance in which, for an *incomplete* constraint, the union of the subclasses is properly contained in the superclass, and for an *overlapping* constraint, at least two subclasses are not disjoint. As already noted in the previous section, this is not required by the UML official semantics. However, this approach can be used for identifying redundant *incomplete* and *overlapping* constraints that cannot be realized in any finite non-empty legal instance.

Example 3.1. The inequality system that ***FiniteSat*** constructs for the class diagram of Figure 1 includes 34 inequalities that are given below. The variables $c,\ r, cr,\ e,\ p,\ d,\ ds,\ ge,\ g$ stand for the classes *Chemical*, *Reaction*, *CatalyzedReaction, Enzyme*, *Protein*, *DNA*, *DNASegment*, *Gene* and *Genome*, respectively. The variables $s,\ ca,\ i,\ co$ stand for the associations *substrate*, *catalyzed*, *include* and *comprise*, respectively.

— **Multiplicity constraint inequalities**:
$s \geq r,\ s \leq r,\ s \geq c$;
$ca \geq 2cr,\ ca \leq 2cr,\ ca \geq e,\ ca \leq e$;

$co \geq ds,\ co \leq ds,\ co \geq g,\ co \leq g$;
$i \geq ds,\ i \leq ds,\ i \geq d$.

— **Class hierarchy constraint inequalities**:
$c \geq p,\ c \geq d,\ cr \geq r,\ p \geq e,\ ds \geq ge,\ ds \geq g$.

— **Generalization set constraints**:
$ds = ge + g$.

— **Non-emptiness inequalities**:
$c,\ r,\ cr,\ e,\ p,\ d,\ ds,\ ge,\ g > 0$ and $s,\ ca,\ i,\ co \geq 0$.

The system is unsolvable for two different reasons:

(1) Inequalities $s \geq r,\ s \leq r$ and $s \geq c$ imply $r \geq c$ and inequalities $ca \geq 2cr,\ ca \leq 2cr,\ ca \geq e$ and $ca \leq e$ imply $e = 2cr$. Together with inequalities $c \geq p$, $p \geq e$, and $cr \geq r$, the system implies $c \geq 2c$ while $c > 0$. These inequalities correspond to the association cycle in Figure 1, as described in Section 1.
(2) Inequalities $co \geq ds,\ co \leq ds,\ co \geq g$ and $co \leq g$ imply $ds = g$. Together with inequality $ds = ge + g$ the system implies $ge = 0$ while $ge > 0$.

Comparison with the algorithm of [Calvanese and Lenzerini 1994] and its optimization in [Cadoli et al. 2004]: For this class diagram (Figure 1), the [Calvanese and Lenzerini 1994] algorithm produces over 12,000 inequalities, and the [Cadoli et al. 2004] optimization reduces this number to over 1000 inequalities. This high number of inequalities results from the high number of new classes and associations that the [Calvanese and Lenzerini 1994] method introduces. Each combination of classes in the original diagram, produces a new, so called *compound* class, leaving out combinations that include a subclass without its super classes. Then, for each association, new, so called *compound* associations, that relate all compound classes that include the original classes of the association, are introduced. The [Calvanese and Lenzerini 1994] system of inequalities includes a variable for every compound class and association. The system has inequalities of two kinds:

(1) **Multiplicity constraint inequalities:** For each original association with association ends with original classes C_1, C_2, and a compound class $\bar{C}$ that includes either C_1 or C_2, the system includes between 0 to 2 inequalities (depending on the multiplicity constraints on the opposite side of the association, where minimum 0 and maximum * do not produce inequalities). For the class diagram in Figure 1, there are above 800 such inequalities.
(2) **Non-negative inequalities:** For every new compound class or association, represented by a variable $\bar{t}$, $\bar{t} \geq 0$. For the class diagram in Figure 1, there are 104 non-negative inequalities for the 104 compound class variables, and over 12,000 such inequalities for compound associations.

The [Cadoli et al. 2004] optimization for the *disjoint-complete* constraint, reduces the number of multiplicity constraint inequalities to over 400 inequalities, the compound class non-negative inequalities to 62, and the compound association non-negative inequalities to over 10,000. The size of the inequality system is further reduced down to around 1,000 inequalities, by applying several optimization heuristics.

3.1. Correctness and Complexity of *FiniteSat*

Proving the correctness of the ***FiniteSat*** algorithm requires an 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 class hierarchy constraints, directed from superclasses to their subclasses (association lines are removed, and GS constraints are considered as plain class hierarchy constraints).

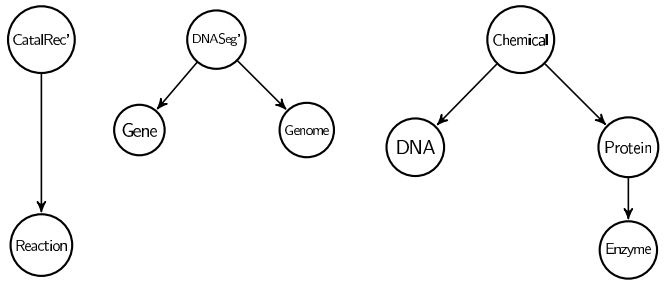


Fig. 6: The class hierarchy graph of Figure 1

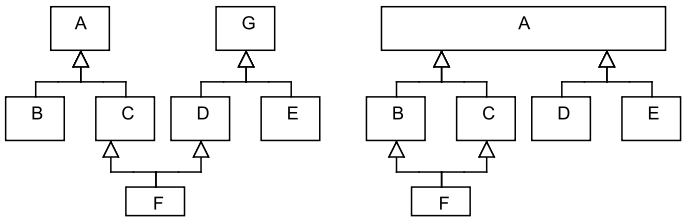


Fig. 7: An acyclic and a cyclic hierarchy structures

We consider two versions of such graphs: Directed and undirected. Figure 6 shows the class hierarchy graph of Figure 1.

Three class hierarchy structures are analyzed:

(1) *Tree class hierarchy*: The directed graph of the class hierarchy forms a set of trees, as in Figure 1.
(2) *Acyclic class hierarchy*[8]: The undirected graph of the class hierarchy is acyclic. In Figure 7-a, the directed class hierarchy is not a set of trees, as F is a subclass of both C and D, but the undirected class hierarchy graph is acyclic (a tree).
(3) *Cyclic class hierarchy*: The undirected graph of the class hierarchy is cyclic. Multiple inheritance is unrestricted, as the undirected induced graph can be cyclic. In Figure 7-b, class F has two class hierarchy paths to its superclass A. The class hierarchy path A, B, F, C, A forms an undirected class hierarchy cycle[9].

The correctness of Algorithm ***FiniteSat*** is proved in two steps:

(1) First, finite satisfiability of a class diagram CD is reduced to finite satisfiability of a class diagram CD', that does not include class hierarchies.
(2) Second, finite satisfiability of CD' is reduced to solvability of the inequality system produced by ***FiniteSat***.

[8]Note that this is an undirected class hierarchy relation, and it differs from the non-circular class hierarchy relationship of UML, as defined in the abstract syntax in Section 2.

[9]The examples presented in the FiniteSatUSE tool site [BGU Modeling Group 2011a] demonstrate the three kinds of class hierarchy structures. The Electronic-interlocking-system examples include only tree class hierarchies. The Academic-domain examples include an acyclic (non-tree) class hierarchy structure – classes *FacultyMember, Employee, Academic*, and cyclic class hierarchy structures – classes *Proposal, EngineeringStudies, NaturalStudies, Specialization* form a cycle, and classes *Graduate, MAGraduate, PhDGraduate, Academic* form another cycle.

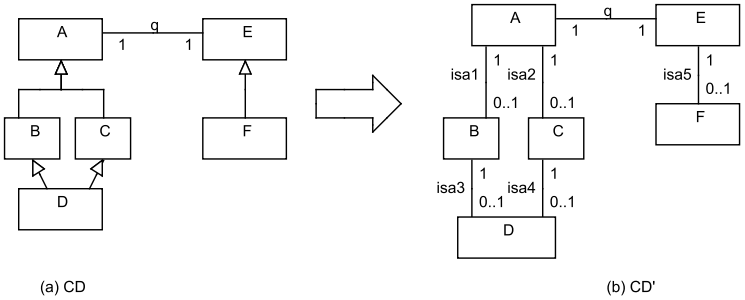


Fig. 8: A class diagram reduction

The reductions depend on the structures of class hierarchies, and the presence of GS constraints. The first step reduction does not hold for class hierarchy structures whose graphs include cycles with *disjoint* or *complete* constraints.

3.1.1. Reduction of Finite Satisfiability of CD to Finite Satisfiability of CD'.

Translation of CD to CD':

(1) Initialize CD' by CD.
(2) Remove every GS constraint $GS(C, C_1, \ldots, C_n;\ const)$, leaving only the n class hierarchy constraints $C_1 \prec C,\ \ldots,\ C_n \prec C$.
(3) 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 superclass end in the diagram) and 0..1 participation constraint for the superclass. Figure 8-b shows the reduced class diagram of Figure 8-a.
(4) For a GS constraint $GS(C, C_1, \ldots, C_n;\ const)$ in CD, if $ISA_1,\ \ldots,\ ISA_n$ are the associations in CD' that replace its n class hierarchy constraints (entry (3) above), insert in CD' a GS constraint *const'* on these ISA associations, as follows:
   (a) *const* = $\langle disjoint \rangle$:
   const' = "every object e of C may participate in exactly one link of the associations $ISA_1,\ \ldots,\ ISA_n$" (a xor-constraint on the ISA associations).
   (b) *const* = $\langle overlapping \rangle$:
   const' = "there exists an object of C that participates in at least two ISA association links".
   (c) *const* = $\langle complete \rangle$:
   const' = "every object of C participates in an ISA association link".
   (d) *const* = $\langle incomplete \rangle$:
   const' = "there exists an object of C that does not participate in any ISA association link".
   If $const$ is a pair constraint, insert in CD' a constraint that combines the constraints of its components.

Note: The ***FiniteSat*** algorithm adopts the strict interpretation for the *overlapping* and *incomplete* constraints, which requires the existence of at least one instance with overlapping and incomplete covering, respectively. The constraint *const'* in CD' reflects the strict semantics.

Mapping instances between CD and CD':

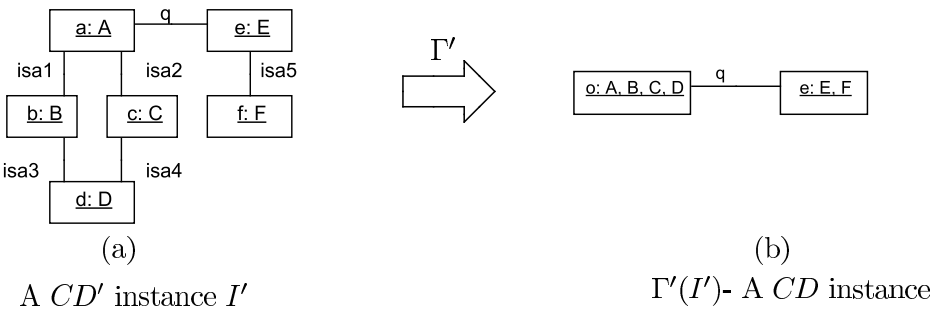


Fig. 9: The Γ' mapping of a CD' instance to a CD instance

(1) Γ **– Mapping an instance** I **of** CD **to an instance** I' **of** CD'**:**
   (a) I' has the semantic domain of I, and the same class and association extensions.
   (b) For every class hierarchy constraint $D \prec C$ in CD (including class hierarchies that are left from GS constraints), and $\mathtt{e} \in D^I$: If the corresponding ISA association is ISA_D, relate $\mathtt{e}$ in I' by ISA_D, i.e., $(\mathtt{e}, \mathtt{e}) \in ISA_D^{I'}$.

(2) Γ' **– Mapping an instance** I' **of** CD' **to an instance** I **of** CD**:**
   (a) Collapse ISA linked objects: For every structure of ISA linked objects $\mathtt{o}_1, \ldots, \mathtt{o}_n$ in I', insert a single new object $\mathtt{o}$ to all classes of the objects in the structure: $\mathtt{o} = \Gamma'(\mathtt{o}_i)$, for $i = 1..n$. Figure 9 demonstrates the Γ' mapping. The intuition is that CD' splits a single instance object of CD into its components in its ancestor classes.
   (b) Populate CD^I classes with the rest of I' objects that are not ISA related: For every $\mathtt{o} \in C^{I'}$, such that $\mathtt{o}$ is not ISA linked: $\Gamma'(\mathtt{o}) = \mathtt{o} \in C^I$.
   (c) Preserve association extensions: For every regular association a in CD', and link $(\mathtt{o}_1, \mathtt{o}_2) \in a^{I'}$, insert the link $(\Gamma'(\mathtt{o}_1), \Gamma'(\mathtt{o}_2))$ into a^I.

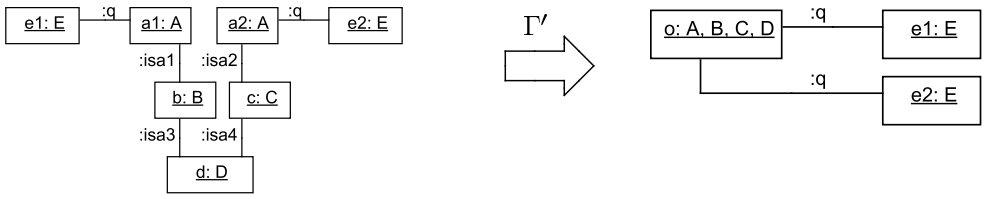
The goal is to show that the above mappings preserve legal instances. That is, if I is a legal instance of CD, then I' is a legal instance of CD', and vice versa. While this is immediate for the Γ mapping, it is not always true for Γ'.

I. Preserving finite satisfiability from CD **to** CD' **– The** Γ **translation:**

CLAIM 3.2. *If CD is finitely satisfiable, then CD' is also finitely satisfiable.*

PROOF. Let I be a non-empty finite legal instance of CD, and denote $I' = \Gamma(I)$. All multiplicity constraints on regular associations and on ISA constraints are satisfied. It remains to show that I' satisfies the corresponding *GS* constraints: For a GS constraint $GS(C, C_1, ...C_n; Const)$ in CD:

(1) *Const* = $\langle disjoint \rangle$: The extensions of $C_1, ...C_n$ in I are pairwise disjoint. Therefore, for an object $\mathtt{e} \in {C_i}^I$, $(\mathtt{e}, \mathtt{e}) \in ISA_i^{I'}$ and for each $j \neq i$, $(\mathtt{e}, \mathtt{e}) \notin ISA_j^{I'}$. Hence the xor-constraint is satisfied.
(2) *Const* = $\langle overlapping \rangle$: There are two classes from $C_1, ...C_n$ that are overlapping in I. If $\mathtt{e} \in {C_i}^I \bigcap {C_j}^I$, then $(\mathtt{e}, \mathtt{e}) \in ISA_i^{I'} \cap ISA_j^{I'}$.
(3) *Const* = $\langle complete \rangle$: An object $\mathtt{e} \in C^I$ is also an object of at least one subclass ${C_i}^I$. Therefore, $(\mathtt{e}, \mathtt{e}) \in {ISA_i}^{I'}$.
(4) *Const* = $\langle incomplete \rangle$: There exists an object $\mathtt{e} \in C^I$ which does not belong to any subclass of C. Therefore, it does not particpate in any ISA-link in I'.



(a) A CD' legal instance I'

(b) $\Gamma'(I')$- An illegal instance I

Fig. 10: The Γ' mapping of a legal CD' instance that does not satisfy the single class property

The proofs for pair constraints are obtained by combining the proofs of the single constraints. □

II. Preserving finite satisfiability from CD' to CD – The Γ' translation:

The Γ' translation might map a legal instance I' of CD' into an illegal instance I of CD. Therefore, it is necessary to characterize legal instances of CD' whose Γ' translation yields a legal CD instance. The *single class property* defined below guarantees that $\Gamma'(I')$ is a legal CD instance, for a legal CD' instance I'.

Definition 3.3. (Single class property) An instance I' of CD' has the *single class property* if every structure of ISA-linked objects does not include two objects from the same class.

The rationale behind this property has to do with the Γ' mapping for ISA related objects. Γ' collapses a structure of ISA related objects into a single object that belongs to all classes of objects in the structure (see Figure 9). The problem is that if several ISA linked objects from the same class in I' collapse into a single object in I, then this new I object has the links of all objects from which it originates, and therefore might violate the multiplicity constraints. Figure 10-a presents a legal instance I' of the class diagram in Figure 8-b that does not satisfy the Single Class Property. Its $\Gamma'(I')$ mapping is an illegal instance of the class diagram in Figure 8-a since object o in class A is q-related to two objects in class E, while the multiplicity constraint dictates exactly one.

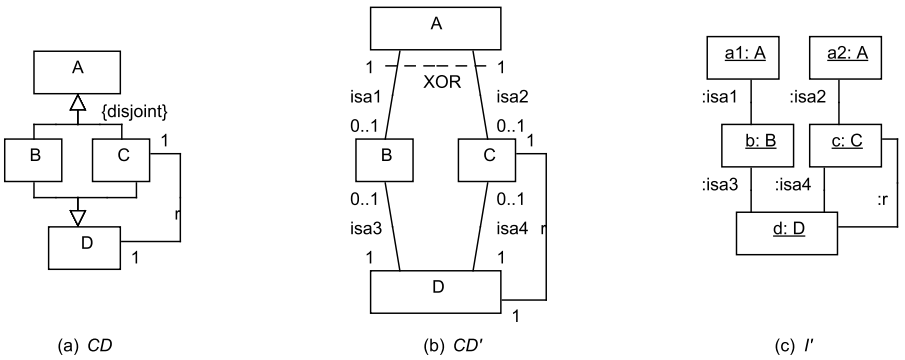
CLAIM 3.4. *If a non-empty finite legal instance I' of CD' satisfies the single class property then $\Gamma'(I')$ is a non-empty finite legal instance of CD.*

PROOF. In the appendix. □

The following claim provides a syntactic characterization of the existence of the single class property. It defines the CD diagrams for which a finitely satisfiable CD' has a legal instance that satisfies the single class property. Together with Claim 3.4 it defines the CD diagrams, for which the Γ and Γ' mappings reduce the finite satisfiability of CD to the finite satisfiability of CD'.

CLAIM 3.5. *(Single class property existence)*

(1) *If CD has a tree or acyclic class hierarchy structure, then every legal instance of CD' satisfies the single class property.*
(2) *If the class hierarchy structure of CD does not include cycles with a* disjoint *or a* complete *constraint, then a finitely satisfiable CD' has a non-empty finite legal*



(a) *CD* (b) *CD′* (c) *I′*

Fig. 11: A Class Diagram CD with a class hierarchy cycle that includes a *disjoint* constraint, for which CD' does not have a non-empty finite legal instance that satisfies the single class property

instance I'' that satisfies the single class property. Moreover, I'' can be constructed from any non-empty finite legal instance, by a procedure that preserves class and association sizes.

PROOF. In the appendix. □

Figures 11-a and 12-a present class diagrams (named CD) that include class hierarchy cycles with a *disjoint* and a *complete* constraints, respectively. Figures 11-b and 12-b present the CD' translations of these class diagrams, and Figures 11-c and 12-c present legal instances of the CD' class diagrams that do not satisfy the single class property. It can be easily verified that every non-empty finite legal instance of the CD' diagrams does not satisfy the single class property[10].

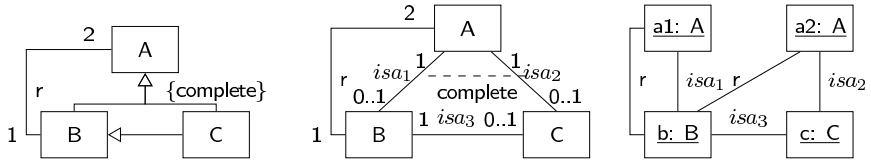
Based on Claims 3.2, 3.4 and 3.5 we get the main result for reducing finite satisfiability between the class diagrams:

THEOREM 3.6. *(Reduction of finite satisfiability between class diagrams). Let $\mathcal{CD}_{M,CH,GS}$ denote class diagrams with multiplicity constraints, class hierarchy and GS constraints, and $\mathcal{CD}_{M,GS}$ denote class diagrams as in the CD' construction, with multiplicity constraints and GS constraints on ISA associations*[11].

(*1*) *Finite satisfiability in $\mathcal{CD}_{M,CH,GS}$ is effectively reducible to finite satisfiability in $\mathcal{CD}_{M,GS}$, for class diagrams in $\mathcal{CD}_{M,CH,GS}$ without class hierarchy cycles that include a* disjoint *or a* complete *constraint.*
(*2*) *Any finitely satisfiable class diagram in $\mathcal{CD}_{M,CH,GS}$ can be effectively translated into a finitely satisfiable class diagram in $\mathcal{CD}_{M,GS}$.*

[10]Similar structures occur in the examples in the FiniteSatUSE site [BGU Modeling Group 2011a]. In the Molecular-biology example, the CD' translation of the sub-diagram created by classes *Molecule, SmallMolecule, MacroMolecule, Compound* does not have a non-empty finite legal instance that satisfies the single class property, implying that the same holds for the entire diagram. In the Academic-domain examples, the sub-diagram *Proposal, EngineeringStudies, NaturalStudies, Specialization, GraduateStudy, Field, Certificate, Graduate* has the same effect.

[11]This theorem implies that in presence of GS constraints, class hierarchy constraints cannot be removed by replacing them with constrained associations, as in the CD to CD' translation presented in this subsection, and in contrast to the claim of [Gogolla and Richters 2002].



(a) CD (b) CD' (c) I'

Fig. 12: A Class Diagram CD with a class hierarchy cycle that includes a *complete* constraint, for which CD' does not have a non-empty finite legal instance that satisfies the single class property

COROLLARY 3.7. *If a class diagram CD in $\mathcal{CD}_{M,CH,GS}$ is translated into a non-finitely satisfiable class diagram in $\mathcal{CD}_{M,GS}$, then CD is also non-finitely satisfiable.*

*3.1.2. Reduction of Finite Satisfiability of $\mathcal{CD}_{M,GS}$ to Solvability of the inequality system produced by **FiniteSat**.*

This reduction extends the [Lenzerini and Nobili 1990] reduction that applies only to class diagrams with binary multiplicity constraints. In the direction from class diagrams to the solvability of the inequality system, the extended proof handles the inequalities that reflect the GS constraints. In the opposite direction, the GS constraints dictate a careful construction of a non-empty finite legal instance of CD': At every round of the population of ISA associations on which GS constraints are imposed, the objects of the superclass are reordered, so to guarantee satisfaction of the GS constraints.

CLAIM 3.8. *If CD' is finitely satisfiable, then Ψ^{CD} is solvable.*

PROOF. In the Appendix. □

CLAIM 3.9. *If Ψ^{CD} is solvable, then CD' is finitely satisfiable.*

PROOF. In the Appendix. □

These claims prove the second step of ***FiniteSat*** correctness:

THEOREM 3.10. *(Reduction of Finite Satisfiability of $\mathcal{CD}_{M,GS}$ to inequality solvability). Finite satisfiability in $\mathcal{CD}_{M,GS}$ is reducible to solvability of the inequality system produced by **FiniteSat** .*

Comment: The last result has implication to relational databases. Class diagrams in $\mathcal{CD}_{M,GS}$ single out a special kind of ISA association, with 0..1 and 1..1 multiplicities. Such associations are implemented, in the relational paradigm, using foreign-key attributes, that are themselves key attributes. This is the standard way to implement hierarchy between tables. This result enables to determine finite satisfiability of database schemas that have associated integrity constraints of *disjoint, complete, incomplete, overlapping*, imposed on groups of key foreign-key attributes.

Putting together the results of the two step proof (theorems 3.6 and 3.10), we obtain the correctness theorem for ***FiniteSat***:

THEOREM 3.11. *(**FiniteSat** correctness – Reduction of Finite Satisfiability of $\mathcal{CD}_{M,CH,GS}$ to inequality solvability).*

(*1*) *Finite satisfiability in $\mathcal{CD}_{M,CH,GS}$ is reducible to solvability of a linear inequality system, for class diagrams in $\mathcal{CD}_{M,CH,GS}$ without class hierarchy cycles that in-*

*clude a disjoint or a complete constraint. The reduction is given by the **FiniteSat** algorithm.*

(2) *A finitely satisfiable class diagram in $\mathcal{CD}_{M,CH,GS}$ can be effectively translated by **FiniteSat** into a solvable linear inequality system.*

COROLLARY 3.12. *If the application of **FiniteSat** to a class diagram CD in $\mathcal{CD}_{M,CH,GS}$ returns an unsolvable inequality system, then CD is non-finitely satisfiable.*

Theorem 3.11 draws the scope of the ***FiniteSat*** algorithm. Indeed, for the two class diagrams in Figures 11-a and 12-a, while the class diagrams are not finitely satisfiable, ***FiniteSat*** yields solvable inequality systems[12]. ***FiniteSat*** fails on 11-a since the *disjoint* constraint on the A-GS projects a *disjoint* constraint on the E-GS, but this constraint is not recorded in the inequality system. ***FiniteSat*** fails on 12-a since the class hierarchy $C^I \subseteq B^I$, which implies $B^I \cup C^I = B^I$ is not recorded in the inequality system.

CLAIM 3.13. *(**FiniteSat** Complexity) The construction of the inequalities by **FiniteSat** has $O(n)$ complexity, where n is the number of constraints in the class diagram.*

PROOF. Every constraint contributes a constant number of inequalities. □

*Implementation of **FiniteSat**.* The ***FiniteSat*** algorithm is implemented, within the ***FiniteSatUSE*** tool [BGU Modeling Group 2011b], as an extension of the USE system [Gogolla et al. 2005, 2009]. The USE grammar that defines class diagrams is extended for supporting GS constraints. The implementation uses an off-the-shelf application for solving the system of linear inequalities produced by ***FiniteSat***. By Theorem 3.11 we know that negative answers of ***FiniteSat***, i.e., detection of finite satisfiability problems, are reliable for all class diagrams, while positive answers are reliable only for class diagrams without class hierarchy cycles that include *disjoint* or *complete* constraints (henceforth *problematic class hierarchy cycles*). Therefore, positive answers of ***FiniteSat*** trigger an application that provides a warning if the class diagram includes problematic class hierarchy cycles, and singles out all such cycles.

FiniteSat has been tested on several real class diagrams of reasonable size (about 25 classes). [BGU Modeling Group 2011a] shows some of these class diagrams and the results, including timing. [Makarenkov et al. 2009] describes our experiments with large synthetic class diagrams. We have tested both the frequency of occurrence of the finite satisfiability problem and the scalability of the ***FiniteSat*** algorithm. The results show that with high statistical confidence, the finite satisfiability problem occurs even with low presence of constrained associations, and that ***FiniteSat*** easily scales to large class diagrams (0.77 minutes for class diagrams with 500 classes and 500 constrained associations).

4. PROPAGATION OF *DISJOINT* AND *INCOMPLETE* GS CONSTRAINT

One reason for the limitation of ***FiniteSat*** over class hierarchies that include cycles with *disjoint* or *complete* constraints is that such cycles enable projection of constraints. That is, the cycles might impose constraints that are not stated in the class diagram, and therefore do not appear in the inequality system that ***FiniteSat*** constructs (as in Figure 11-a). In this section we present a method for propagation of

[12] 11-a is not finitely satisfiable since in every legal instance I, $B^I \cap C^I = \emptyset$ and $B^I \cup C^I \subseteq D^I$, while $|D^I| = |C^I|$; 12-a is not finitely satisfiable since in every legal instance I, $C^I \subseteq B^I$ and $B^I \cup C^I = A^I$, while $2|B^I| = |A^I|$

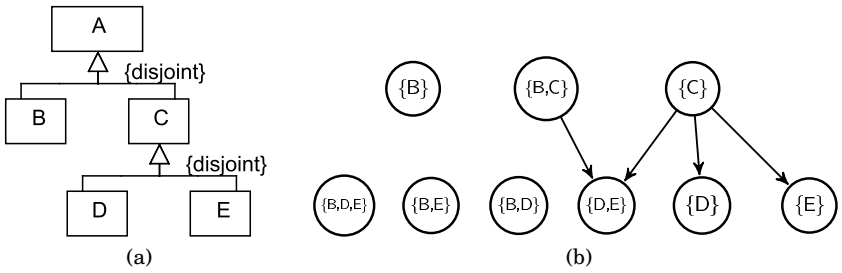


Fig. 13: The set $A^{\prec^* dis}$ of disjoint descendant sets of class A

missing *disjoint* and *incomplete* constraints. We show that the propagation extends the scope of ***FiniteSat***, and suggest it as a preprocessing stage.

Terminology. The class hierarchy relations $\prec$, $\preceq$, $\prec^*, \prec^+$ are extended to sets of classes. For a class A and sets of classes $\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2$, $\mathcal{S} \preceq A$ means that for every class A_i in $\mathcal{S}$, $A_i \preceq A$, and $\mathcal{S}_1 \preceq \mathcal{S}_2$ means that for every class A_1 in $\mathcal{S}_1$ there is a class A_2 in $\mathcal{S}_2$ such that $A_1 \preceq A_2$ (similarly for $\prec, \prec^*, \prec^+$). The *weak descendant set of a class* A, denoted $A^{\prec^*}$, is the set of classes A' such that $A' \prec^* A$; $A^{\prec^+}$ is the set of *descendants of* A ($A' \prec^+ A$).

4.1. Propagation of *disjoint* GS Constraints

Two classes are *disjoint* if their extensions are disjoint in every legal instance of the class diagram. A set $\mathcal{S}$ of descendant classes of a class A that are pairwise disjoint, is called a *disjoint descendant set of* A. The restriction of $\prec^*$ to disjoint descendant sets of a class A is denoted $\prec^{*dis}$. That is, $\mathcal{S} \prec^{*dis} A$ if $\mathcal{S} \prec^* A$ and its member classes are pairwise disjoint. The set of all disjoint descendant sets of a class A, denoted $A^{\prec^* dis}$, is partially ordered by the direct descendant relation $\prec$. Figure 13-b shows the partial ordering of the set $A^{\prec^* dis}$ in Figure 13-a.

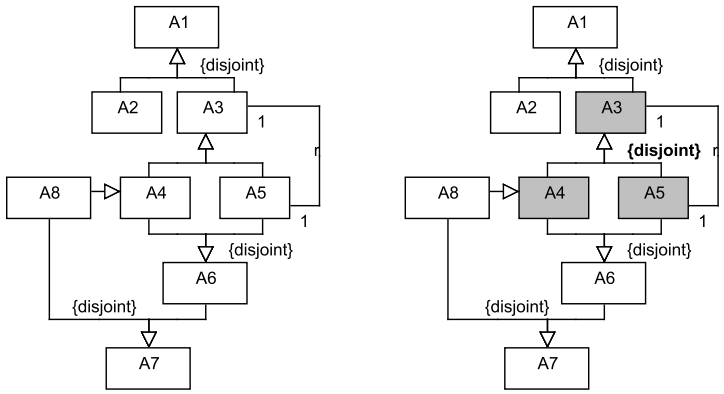
A generalization set G is a *logical implication of a class diagram* CD, denoted $CD \models G$, if $I \models G$ in every legal instance I of CD. In Figure 11-a, in addition to the $GS(A, B, C; \langle disjoint \rangle)$ GS constraint that is explicitly specified, $GS(D, B, C; \langle disjoint \rangle)$ is an implied GS constraint. The following proposition shows that every disjoint descendant set of a class yields an implied GS.

PROPOSITION 4.1. *If* $\{D_1, \ldots, D_n\} \prec^{*dis} A$, *then* $CD \models GS(A, D_1, \ldots, D_n; \langle disjoint \rangle)$.

PROOF. By definition, all classes in $\{D_1, \ldots, D_n\}$ are descendants of A, and are pairwise disjoint. Therefore, in every legal instance of the class diagram, the constraint $GS(A, D_1, \ldots, D_n; \langle disjoint \rangle)$ holds. □

This proposition states that extending a class diagram with GS constraints for the disjoint descendant sets of its classes does not change its meaning: The extended class diagram has the same set of legal instances. The important point is that while the meaning of the class diagram is not affected, the following claim and example show that such extensions might strengthen the performance of ***FiniteSat***: The algorithm stays sound for non-finite satisfiability, and in addition, can detect finite satisfiability problems that are not detected in the original class diagram.

CLAIM 4.2. *(Extending CD with implied disjoint GS constraints) Let CD be a class diagram, and let $\overline{CD}$ be CD, augmented with a GS constraint $GS(A, D_1, \ldots, D_n; \langle disjoint \rangle)$, for all $\{D_1, \ldots, D_n\}$ in $A^{\prec^* dis}$, for some class A in CD. Then:*



(a) CD (b) pruned $\overline{CD}$

Fig. 14: A class diagram CD and a pruned propagation of *disjoint* GS constraints

(*1*) $CD \equiv \overline{CD}$; [13]
(*2*) *Every solution of* $\Psi^{\overline{CD}}$ *is a solution of* Ψ^{CD}*, but not necessarily vice versa.*

PROOF. Class diagram equivalence results from Proposition 4.1. The inequality systems relation holds since $\Psi^{\overline{CD}}$ includes Ψ^{CD}, and Example 4.3 shows that not all Ψ^{CD} solutions solve $\Psi^{\overline{CD}}$ (implying that ***FiniteSat*** identifies more finite satisfiability problems when applied to $\overline{CD}$). □

Example 4.3. (Strengthened ***FiniteSat***) Consider Figure 11-a. Algorithm ***FiniteSat*** cannot detect the finite satisfiability problem. But, when the class diagram is augmented with $GS(D, B, C; \langle disjoint \rangle)$ (note that $\{B, C\} \prec^{*dis} D$), ***FiniteSat*** adds the new inequality $d \geq b + c$, which accounts for the new *disjoint* constraint, and turns the whole inequality system unsolvable.

4.1.1. Redundancy of implied disjoint GS *constraints.*

In order to use the result of Claim 4.2, one has to augment a class diagram with GSs for disjoint descendant sets of classes. It is not feasible to apply this extension to all such GSs, for all classes, since there are too many such sets (exponential in the sizes of disjoint GSs and class hierarchy depths).

A closer consideration of the *disjoint* GSs in $\overline{CD}$ reveals that not all are needed, since they do not strengthen ***FiniteSat***. We say that a *disjoint* GS constraint G in $\overline{CD}$ is *redundant* if removing it from $\overline{CD}$ does not weaken ***FiniteSat***. That is, $\Psi^{\overline{CD}} \equiv \Psi^{\overline{CD}-G}$. G can be identified as redundant if it is *covered* by another *disjoint* GS constraint G', i.e., the inequalities that ***FiniteSat*** introduces for G' imply those introduced for G. For example, in Figure 14-a, the implied constraint $GS(A_1, A_2, A_4, A_5; \langle disjoint \rangle)$ is covered by $GS(A_1, A_2, A_3; \langle disjoint \rangle)$, while the implied constraint $GS(A_3, A_4, A_5; \langle disjoint \rangle)$, is not covered by any other constraint. We observe two cases of coverage, due to subclass and to superclass coverage.

Redundancy due to subclass coverage. A set $\mathcal{S} \in A^{\prec^* dis}$ is *covered* by a set $\mathcal{S}' \in A^{\prec^* dis}$, if $\mathcal{S} \prec^* \mathcal{S}'$, and $\mathcal{S} \neq \mathcal{S}'$. For example, in Figure 14-a, the set $\{A_2, A_8\}$ is *covered* by $\{A_2, A_4\}$, which is *covered* by $\{A_2, A_4, A_5\}$, which is *covered* by $\{A_2, A_3\}$. The *cover*

[13] Class diagram equivalence is defined in Section 2, in the Semantics paragraph.

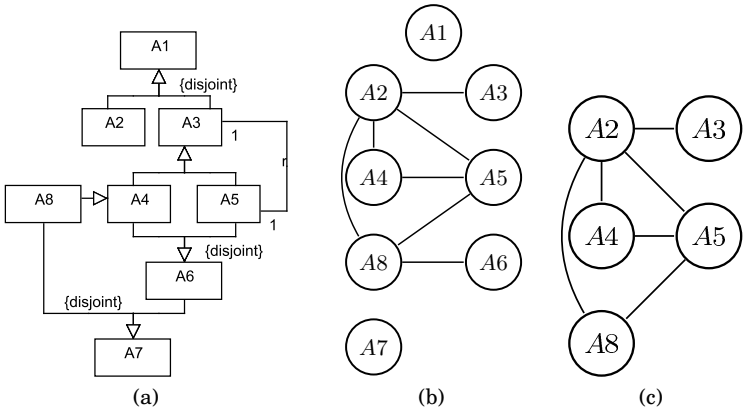


Fig. 15: The disjoint graph of a class diagram and of class A_1

relation is a partial order on $A^{\prec^* dis}$. The following proposition characterizes redundant *disjoint* GS constraints in $\overline{CD}$ as the ones having a *cover*.

PROPOSITION 4.4. *Let G be a disjoint GS constraint for a superclass A, with a set of subclasses $\mathcal{S}$. If $\mathcal{S}$ is covered by another set in $A^{\prec^* dis}$, G is redundant. That is, removing it from $\overline{CD}$ does not weaken **FiniteSat**.*

PROOF. In the Appendix. □

The last proposition states that for a class A, non-redundant implied *disjoint* GS constraints are those whose sets of subclasses are maximal elements of the *cover* relation on $A^{\prec^* dis}$.

Redundancy due to superclass coverage. A constraint $G = GS(C, C_1, \ldots, C_n; disjoint)$ is covered by a constraint $G' = GS(C', C_1, \ldots, C_n; disjoint)$ where $C' \prec^+ C$, since the inequalities that ***FiniteSat*** introduces for G' are $c' \geq \sum_{i=1}^{n} c_i$, which together with the class hierarchy inequalities $c \geq \ldots \geq c'$ imply $c \geq \sum_{i=1}^{n} c_i$. Therefore, G is redundant, and removing it from $\overline{CD}$ does not weaken ***FiniteSat***.

Altogether, $\overline{CD}$ can be meaningfully pruned, without weakening ***FiniteSat***. Figure 14-b shows a pruned $\overline{CD}$ for Figure 14-a.

4.1.2. Constructing the pruned $\overline{CD}$:.

In order to compute a pruned $\overline{CD}$ we need to find, for a class A, the maximal elements of its *cover* relation on $A^{\prec^* dis}$ (the ordered set of its disjoint descendants). For that purpose we introduce the notion of a *disjoint graph* of CD, denoted $dis\_graph(CD)$, which is an undirected graph whose nodes are the classes of CD, and its edges connect nodes of disjoint classes. For example, a disjoint graph of Figure 15-a is given in Figure 15-b. The restriction of $dis\_graph(CD)$ to the set of descendants of a class A is denoted $dis\_graph(A^{\prec^*})$, and is defined as $dis\_graph(CD)_{/A^{\prec^*}}$. The graph in 15-c is the restriction of 15-b to the descendant set of class A_1.

Given a $dis\_graph(A^{\prec^*})$ for some class A, its cliques are sets of pairwise disjoint descendants of A, and therefore belong to $A^{\prec^* dis}$. Maximal cliques that are top elements in the $\preceq$ ordering of $A^{\prec^* dis}$, are maximal elements of the *cover* relation on $A^{\prec^* dis}$. By Proposition 4.4, these are the subclasses in the non-redundant *disjoint* GS constraints for A. Algorithm ***Disjoint Propagation*** (Algorithm 2) uses this analysis in order to

ALGORITHM 2: *Disjoint Propagation*

Input: A class diagram CD, and its $dis\_graph(CD)$.

Output: A pruned class diagram $\overline{CD}$, that augments CD with non-redundant *disjoint* GS constraints.

begin

for *For all classes* A *in* CD **do**

(1) Create $dis\_graph(A^{\prec^*})$.

(2) Select the maximal cliques in $dis\_graph(A^{\prec^*})$.

(3) Order the maximal cliques by $\preceq$.

(4) Select the top elements in the $\preceq$ ordering of the maximal cliques.

(5) For every selected set $A_1, \ldots, A_n$ of disjoint descendants of A:

(a) If $GS(A, A_1, \ldots, A_n; disjoint)$ is not covered by an already added constraint in CD (superclass coverage), add it to CD.

(b) Remove from CD all previously added *disjoint* GS constraints that are covered by the new GS.

end

end

compute a pruned $\overline{CD}$, for a given $dis\_graph(CD)$. Later on we discuss the issue of $dis\_graph(CD)$ construction.

Example 4.5. Application of ***Disjoint Propagation*** to the class diagram in Figure 15-a, and its disjoint graph in Figure 15-b:

(1) **Class** A_1:
— Step 1: Create the graph $dis\_graph(A_1^{\prec^*})$, presented in Figure 15-c: $(dis\_graph(CD)_{/\{A_2,A_3,A_4,A_5,A_8\}})$.
— Steps 2, 3, 4: The maximal cliques in $dis\_graph(A_1^{\prec^*})$ are $\{A_2, A_3\}$, $\{A_2, A_4, A_5\}$, $\{A_2, A_5, A_8\}$. The single top element is $\{A_2, A_3\}$.
— Step 5: The GS constraint $GS(A_1, A_2, A_3; \langle disjoint \rangle)$ is already in CD.
(2) **Classes** A_2, A_5, A_8 have no descendants.
(3) **Class** A_4: Has a single descendant.
(4) **Class** A_3:
— Step 1: $dis\_graph(A_3^{\prec^*}) = dis\_graph(CD)_{/\{A_4,A_5\}}$
— Steps 2, 3, 4: $dis\_graph(A_3^{\prec^*})$ includes a single clique $\{A_4, A_5\}$.
— Step 5: Add the GS constraint: $GS(A_3, A_4, A_5; \langle disjoint \rangle)$.
(5) **Class** A_6: Same as class A_3. But no new GS constraint.
(6) **Class** A_7:
— Step 1: $dis\_graph(A_7^{\prec^*}) = dis\_graph(CD)_{/\{A_4,A_5,A_6,A_8\}}$.
— Steps 2, 3, 4: The maximal cliques in $dis\_graph(A_7^{\prec^*})$ are $\{A_4, A_5\}$, $\{A_5, A_8\}$, $\{A_6, A_8\}$. The single top element is $\{A_6,\ A_8\}$.
— Step 5: No new GS constraint.

The class diagram in Figure 15-a is not finitely satisfiable since the extension of A_3 properly includes the extension of A_5, while its members are mapped in a 1:1 manner to members of the extension of A_5. Yet, Algorithm ***FiniteSat*** creates a solvable inequality system. The ***Disjoint Propagation*** preprocessing extends CD with the GS: $GS(A_3,\ A_4,\ A_5; \langle disjoint \rangle)$ (Figure 14-b). For this constraint, ***FiniteSat*** creates the inequality: $a_3 \geq a_4 + a_5$, which contradicts the inequalities $r = a_3$, $r = a_5$ and $a_4 > 0$ (variables a_i stand for classes A_i and r stands for the association r). Therefore, the inequality system is unsolvable, and the pre-processing indeed strengthens ***FiniteSat***.

*Complexity of **Disjoint Propagation**.* The algorithm applies the following operations:

(1) **Disjoint graphs construction:** For all classes, intersect $dis\_graph(CD)$ with $A^{\prec^*}$. This is polynomial in the class diagram size.
(2) **Maximal clique computation:** Cliques computation is known to be exponential in the graph size. Yet, the depth of class hierarchies tend to be bounded – usually not greater than 4, and the size of GS constraints is usually quite small. Consequently, the complexity of maximal clique computation is bounded (exponential in some fixed estimated size of class hierarchies), and independent of the class diagram size.
(3) **Top clique selection:** Polynomial in the size of the class diagram.

Altogether, the complexity of ***Disjoint Propagation*** is polynomial in the class diagram size. In any case, Claim 4.2 guarantees that any partial extension of CD by logically implied *disjoint* GS constraints is correct. Therefore, the application of Algorithm ***Disjoint Propagation*** can be dictated by feasibility considerations.

Construction of a disjoint graph of the class diagram. Class disjointness is a semantic notion, that depends on interaction of constraints. Disjoint relations that derive from interaction of *disjoint* GS constraints and class hierarchies can be identified by repeatedly scanning subclasses of disjoint classes. For example, the disjoint graph presented in Figure 15-b can be computed by the following iterations:
Iteration 1: $disjoint(A_2, A_3)$, $disjoint(A_4, A_5)$, $disjoint(A_6, A_8)$. These are the explicitly given *disjoint* constraints.
Iteration 2: $disjoint(A_2, A_4)$, $disjoint(A_2, A_5)$, $disjoint(A_5, A_8)$. $disjoint(A_4, A_8)$ is not computed, since there is a $\prec^*$ relation between them (and therefore are not disjoint).
Iteration 3: $disjoint(A_2, A_8)$.
This step requires polynomial time in terms of the size of the class diagram.

Nevertheless, this simple construction ignores disjoint relations that are caused by interaction with other constraints. Unfortunately, we do not know how to construct the complete disjoint graph, that includes all disjoint relations. Therefore, it might be the case that the complete disjoint graph includes finer top cliques than the ones detected by Algorithm ***Disjoint Propagation***. As a result, the algorithm might insert some *disjoint* GS redundancy, or fail to detect an implied *disjoint* GS.

4.2. Propagation of *disjoint* and *incomplete* GS Constraints

Consider Figure 16-a. The class diagram is not finitely satisfiable since the multiplicity constraints on r and q imply that in every legal instance I, $|{A_1}^I| = |{A_4}^I| + |{A_7}^I|$, ${A_4}^I \bigcap {A_7}^I = \emptyset$, and $A_1 \supset A_4 \cup A_7$. Preprocessing by ***Disjoint Propagation*** adds the $GS(S_1, A_4, A_7; \langle disjoint \rangle)$ constraint, that contributes the $a_1 \geq a_4 + a_7$ inequality. This inequality does not contradict the inequalities that the r and q associations contribute ($r = a_1$, $r = 2a_4$, $q = a_1$, $q = 2a_7$), and therefore, ***FiniteSat*** fails to detect the finite satisfiability problem. The added GS constraint lacks the *incomplete* specification, which would have sharpened the contributed inequality into $a_1 > a_4 + a_7$, thereby causing contradiction.

In this subsection we extend the previous propagation method to account for the interaction between *disjoint* and *incomplete* constraints within class hierarchy cycles. A set of classes $\mathcal{S}$ is an *incomplete disjoint descendant set* of a class A, denoted $\mathcal{S} \prec^{*dis,inc} A$, if $\mathcal{S} \prec^{*dis} A$ and in some legal instance I, $\bigcup_{S' \in \mathcal{S}} S'^I \subset A^I$. The set of all incomplete disjoint descendant sets of a class A, is denoted $A^{\prec^* dis,inc}$.

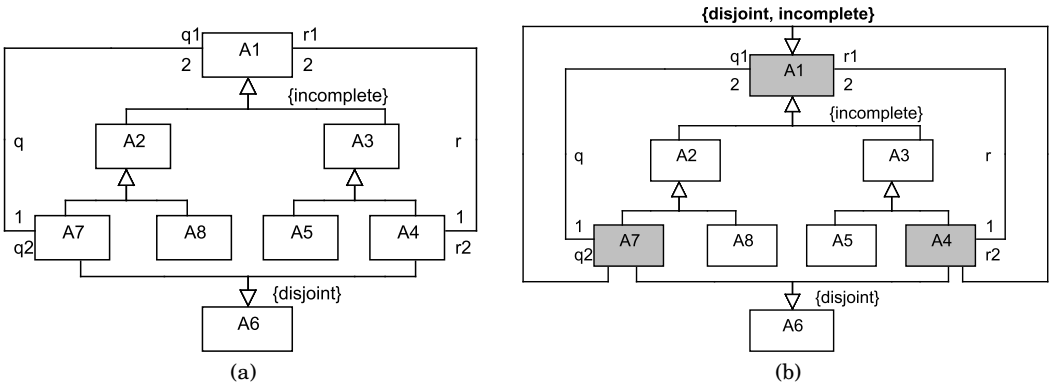


Fig. 16: Propagation of *disjoint* and *incomplete* GS constraints

PROPOSITION 4.6.
If $\{D_1, \ldots, D_n\} \prec^{*dis,inc} A$*, then* $CD \models GS(A, D_1, \ldots, D_n; \langle disjoint, incomplete \rangle)$.

Similarly to the *disjoint* propagation, we first show that extending a class diagram with logically implied *incomplete disjoint* GS constraints can further strengthen ***FiniteSat*** (Claim 4.7). Then, for a class A, we characterize sets in $A^{\prec^* dis}$ which are also in $A^{\prec^* dis,inc}$, and use it to extend Algorithm ***Disjoint Propagation*** with propagation of *disjoint incomplete* GS constraints (Algorithm 3).

ALGORITHM 3: ***Disjoint_Incomplete Propagation***

Input: A class diagram CD, and a $dis\_graph(CD)$.

Output: A pruned class diagram $\overline{CD}^{inc}$, that augments CD with non-redundant *disjoint* GS constraints, and with *disjoint* and *incomplete* GS constraints.

begin
  for *all classes* A *in* CD **do**
    (1) Create $dis\_graph(A^{\prec^*})$.
    (2) Selection of elements from $A^{\prec^* dis,inc}$:
      (a) Choose and order by $\preceq$, cliques whose members are descendants of the set of subclasses of an *incomplete* GS constraint on A.
      (b) Select the top elements in the $\preceq$ ordering of the set of *incomplete* cliques.
      (c) Extend CD with *disjoint, incomplete* GS constraints for A, for all selected sets of *disjoint incomplete* descendants of A.
    (3) Selection of non-redundant elements from $A^{\prec^* dis}$: Apply steps 2 – 5 of Algorithm 2.
  end
end

CLAIM 4.7. *(Extending* CD *with implied disjoint incomplete* GS *constraints) Let* CD *be a class diagram, and let* $\overline{CD}^{inc}$ *be* CD*, augmented for a class* A *in* CD*, with* GS *constraints as follows:*

(1) $GS(A, D_1, \ldots, D_n; \langle disjoint, incomplete \rangle)$*, where* $\{D_1, \ldots, D_n\}$ *is in* $A^{\prec^* dis,inc}$*;*
(2) $GS(A, D_1, \ldots, D_n; \langle disjoint \rangle)$*, for all other* $\{D_1, \ldots, D_n\}$ *in* $A^{\prec^* dis}$*.*

Then:

(*1*) $\overline{CD} \equiv \overline{CD}^{inc}$;
(*2*) *Every solution of* $\Psi^{\overline{CD}^{inc}}$ *is a solution of* $\Psi^{\overline{CD}}$*, but not necessarily vice versa.*

PROOF. Class diagram equivalence results from Proposition 4.6. The inequality systems relation holds since the inequalities for *disjoint incomplete* GS constraints in $\Psi^{\overline{CD}^{inc}}$ imply the inequalities in $\Psi^{\overline{CD}}$, for the *disjoint* constraints, but not vice versa. □

The following proposition characterizes *disjoint* descendant sets that are also *incomplete*:

PROPOSITION 4.8. *If* $\mathcal{S} \in A^{\prec^* dis}$*, and for some incomplete* GS *constraint with superclass* A *and a set of subclasses* $\mathcal{S}'$*,* $\mathcal{S} \prec^* \mathcal{S}'$*, then* $\mathcal{S} \in A^{\prec^* dis,inc}$.

PROOF. Immediate from the definition of $A^{\prec^* dis}$ and the semantics of *incomplete* GS constraints. □

Proposition 4.8 suggests a method for finding sets in $A^{\prec^* dis,inc}$: Find cliques that are descendants of *incomplete* GS constraints on A in CD. Note that this method involves all cliques, not only maximal, since they are not redundant with respect to the *incomplete* GS constraint.

Example 4.9. Application of ***Disjoint_Incomplete Propagation*** to the class diagram in Figure 16-a, and a disjoint graph that includes a single edge between A_4 and A_7:

(1) **Class** A_1:
   — Step 1: $dis\_graph(A_1^{\prec^*})$ has the single edge between A_4 and A_7.
   — Step 2: The only clique in $dis\_graph(A_1^{\prec^*})$ is $\{A_4, A_7\}$. It is a descendant of the set of subclasses of $GS(A_1, A_2, A_3; \langle incomplete \rangle)$. The constraint $GS(A_1, A_4, A_7; \langle disjoint, incomplete \rangle)$ is added to CD.
   — Step 3: Involves the clique $\{A_4, A_7\}$, for which a GS was already add in the previous step.
(2) **Classes** $A_2, A_3, A_4, A_5, A_7, A_8$: Their disjoint graphs are empty (no edges) and therefore they do not add new constraints.
(3) **Class** A_6: Its disjoint graph includes the single $\{A_4, A_7\}$ clique, for which CD includes an explicit *disjoint* constraint.

The output class diagram is shown in Figure 16-b, for which ***FiniteSat*** detects the finite satisfiability problem.

Implementation of ***disjoint Propagation*** *and of* ***Disjoint_Incomplete Propagation*** . The algorithms are implemented, within the ***FiniteSatUSE*** tool [BGU Modeling Group 2011b], as a pre-processing stage to the implementation of ***FiniteSat***. That is, the propagation pre-processor extends the input class diagram with additional *disjoint* and *disjoint incomplete* GS constraints, and ***FiniteSat*** is given the richer class diagram as input. [BGU Modeling Group 2011a] shows the results of applying the ***Disjoint Propagation*** algorithm to several real class diagrams of reasonable size (about 25 classes), including timing. Note that in these examples, class hierarchy depth is at most 4.

5. CAUSE IDENTIFICATION OF FINITE SATISFIABILITY PROBLEMS

The ***FiniteSat*** algorithm presented in Section 3 detects problems of finite satisfiability. However, in a large class diagram, it is desirable to have a more focused information on

which constraints cause the problem (as the counterexamples provided by Alloy [Jackson 2002, 2006] or the explanations provided by AuRUS [Queralt et al. 2010]). In this section we present such a method. Following [Lenzerini and Nobili 1990], the method is based on construction of an identification graph and identifying sub-graphs that point to finite satisfiability problems. This way, once a finite satisfiability problem is detected by the ***FiniteSat*** algorithm, we can point to its cause. The cause identification method has already led to the design of a catalog of anti-patterns for class diagrams, that characterize typical cases of non-finite satisfiability situations, and provide advice for repair [BGU Modeling Group 2010b].

ALGORITHM 4: *Identification Graph* Construction

Input: A class diagram CD with binary multiplicity constraints, class hierarchy constraints, and GS constraints.
Output: A directed graph, $graph(CD)$, with weighted edges.
begin

(1) Create nodes for all classes.
(2) For every multiplicity constraint $r(rn_1 : A_1[min_1, max_1],\ rn_2 : A_2[min_2, max_2])$ connect the nodes A_1, A_2 that correspond to classes A_1, A_2, respectively, by an edge labeled $\frac{max_2}{min_1}$ directed from A_1 to A_2, and an edge labeled $\frac{max_1}{min_2}$ directed from A_2 to A_1. If min_i is 0 or max_i id *, then the label is ∞.
(3) For every class hierarchy constraint $B \prec A$, connect the nodes A, B that correspond to classes A, B, respectively, by an edge labeled 1 from A to B and an edge labeled ∞ from B to A. A GS constraint $G = GS(C,\ C_1,\ \ldots,\ C_n; const)$, $graph(G)$ implies n class hierarchy constraints $C_i \prec C$, for $i = 1..n$.

end

Edges in $graph(CD)$ correspond to multiplicity or class hierarchy constraints in CD, and sub-graphs correspond to sub-diagrams of CD. For a sub-graph G, its corresponding sub-diagram is denoted $CD_{/G}$. It includes all classes and constraints that correspond to nodes and edges in G. Henceforth, for the sake of simplicity, nodes in $graph(CD)$ are identified with the CD classes that they represent.

The following proposition provides the link between finite legal instances of a class diagram to its graph. It enables identification of finite satisfiability problems by sub-graph patterns in the graph of a class diagram. [Lenzerini and Nobili 1990] proves a similar property for its identification graph.

PROPOSITION 5.1. *For every multiplicity constraint* $a(rn_1 : A_1[min_1, max_1],\ rn_2 : A_2[min_2, max_2])$ *in* CD*, edge* $e = \langle A_1, A_2 \rangle$ *from* A_1 *to* A_2 *in* $graph(CD)$ *that corresponds to* a*, and a semantic mapping* I *to* A_1 *and* A_2*:* $\frac{|A_2^I|}{|A_1^I|} \leq label(e)$ *if and only if* I *can be extended to satisfy the* max_2 *and* min_1 *constraints.*

PROOF. *If direction:* The semantics of the multiplicity constraint implies the inequalities: $|A_1^I| \cdot min_2 \leq |a^I| \leq |A_1^I| \cdot max_2$ and $|A_2^I| \cdot min_1 \leq |a^I| \leq |A_2^I| \cdot max_1$. Therefore, $\frac{|A_2^I|}{|A_1^I|} \leq \frac{max_2}{min_1} = label(e)$.

Only-if direction: $\frac{|A_2^I|}{|A_1^I|} \leq label(e) = \frac{max_2}{min_1}$ implies $|A_2^I| \cdot min_1 \leq |A_1^I| \cdot max_2$. The extension a^I can be constructed to satisfy the min_1 and max_2 constraints, following the construction in the proof of Claim 3.5. □

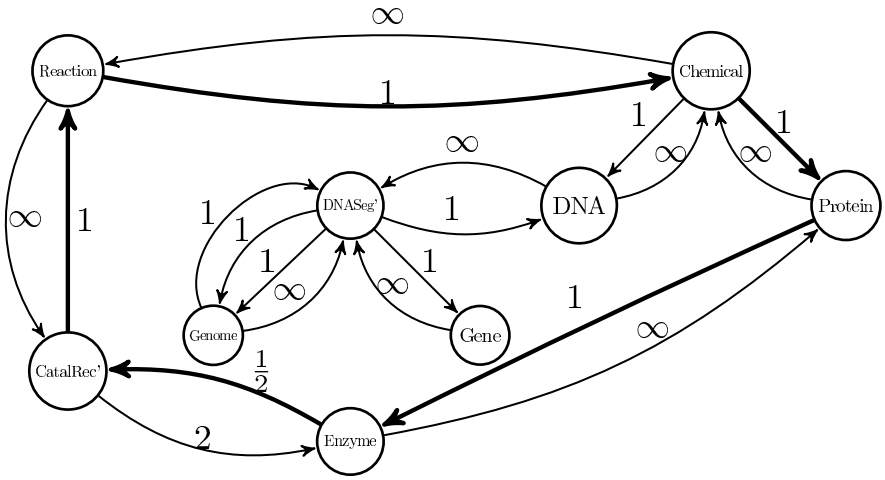


Fig. 17: The identification graph of Figure 1

This proposition implies that for every non-empty finite legal instance I of CD the inequality $\frac{|A_2^I|}{|A_1^I|} \leq label(e)$ holds for all edges in $graph(CD)$. The *weight* of a path in $graph(CD)$ is the multiplication of its edge labels. The last proposition can be generalized for paths, and implies that for every path π from A_1 to A_2 in $graph(CD)$, and every non-empty finite legal instance I for CD: $\frac{|A_2^I|}{|A_1^I|} \leq weight(\pi)$.

A set of constraints is *finitely satisfiable* if there is a finite non-empty instance in which the constraints hold. Otherwise, the set is *not finitely satisfiable*. The sub-graphs of $graph(CD)$, that are identified in this section correspond to sets of non-finitely satisfiable constraints in CD. *Critical cycles* identify problems caused by interaction of multiplicity constraints or interaction of multiplicity and class hierarchy constraints. *Critical sets of GS paths* identify problems caused by interaction of multiplicity or class hierarchy constraints with GS constraints.

5.1. Identification by Critical Cycles

A cycle in $graph(CD)$ whose weight is less than one is called a *critical cycle*. The following claim shows that critical cycles are sub-graphs that identify finite satisfiability problems:

CLAIM 5.2. *Let CD be a class diagram and γ a cycle in $graph(CD)$. Then, $CD_{/\gamma}$ is not finitely satisfiable if and only if γ is critical.*

PROOF. In the Appendix. □

Based on this claim, identification of a critical cycle in $graph(CD)$ points to a non-finitely satisfiable set of multiplicity or class hierarchy constraints.

Example 5.3. Consider the class diagram of Figure 1. Its identification graph, shown in Figure 17, includes the critical cycle $\langle Chemical,\ Protein,\ Enzyme,\ CatalizedReaction,\ Reaction,\ Chemical\rangle$ (shown in bold), which singles out a non-finitely satisfiable set of multiplicity and class hierarchy constraints.

Note: As pointed by [Lenzerini and Nobili 1990], discovering critical cycles can be done in polynomial time, for example by using a variant of the Floyd-Warshall algorithm for

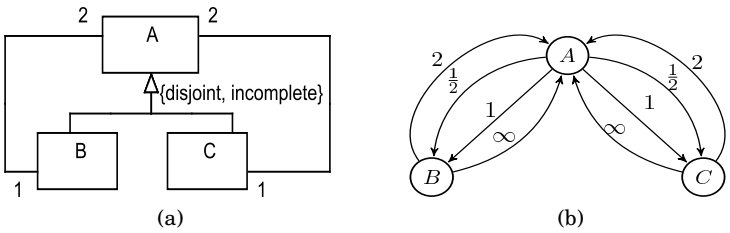


Fig. 18: A class diagram with its identification graph

determining the shortest path between two nodes in a graph. Note also that critical cycles can be turned into *negative cycles*, which are standard in graph theory, by replacing edge labels by their logs, and defining a path weight as the sum of its edge labels.

The following theorem shows that critical cycles provide complete identification of finite satisfiability problems in class diagrams with multiplicity and class hierarchy constraints alone. It is based on a similar result of [Lenzerini and Nobili 1990].

THEOREM 5.4. *A class diagram CD with multiplicity and class hierarchy constraints alone is not finitely satisfiable if and only if $graph(CD)$ includes critical cycles.*

PROOF. In the Appendix. □

Implementation. The ***FiniteSat*** implementation in the ***FiniteSatUSE*** tool [BGU Modeling Group 2011b] is extended by critical cycle identification, based on the ***Identification Graph* Construction** (Algorithm 4). The tool identifies all critical cycles, based on user request (as in the identification of problematic class hierarchy cycles). In the future, we plan to improve the critical cycle identification by employing heuristic strategies that give precedence to identification of small critical cycles that intersect larger critical cycles (e.g., the feedback arc elimination heuristic of [Hartmann 2001]). Altogether, ***FiniteSatUSE*** provides propagation of *disjoint, incomplete* GS constraints, detection of finite satisfiability followed by identification of problematic class hierarchy cycles in case of a positive result, and followed by identification of critical cycles in case of a negative result.

5.2. Identification by Critical Sets of GS Paths

Critical cycles do not identify finite satisfiability problems caused by GS constraints. For example, Figure 18-a presents a non-finitely satisfiable class diagram (the relevant inequalities produced by ***FiniteSat*** are $a > b+c,\ a = 2b,\ a = 2c$, which have no positive solution) whose identification graph, in Figure 18-b, does not include a critical cycle. Nevertheless, the set of two paths $\langle B,\ A\rangle$ and $\langle C,\ A\rangle$ (shown in bold), forms a *critical set of GS paths* (defined below), that identifies a finite satisfiability problem.

For a GS constraint $G = GS(C,\ C_1,\ \ldots,\ C_n; const)$, a G ***path*** is a path in $graph(CD)$ between the nodes C and some C_i. A GS path is *to-super* if it is directed from the subclass node C_i to the superclass node C, and is *to-sub* if it is directed to the subclass node.

The following definitions characterize *critical sets of GS paths*, i.e., sets that identify finite satisfiability problems, as proved in Claim 5.8 below. The characterizations vary by the constraints.

DEFINITION 5.5. *Const* = $\langle disjoint\rangle$ *or* $\langle disjoint, complete\rangle$ *or* $\langle disjoint, incomplete\rangle$*: For a GS constraint $G = GS(C,\ C_1,\ \ldots,\ C_n; const)$, a set of n to-super G paths $\{\pi_i\}_{i=1}^{n}$, one for each subclass C_i, is* critical *if*

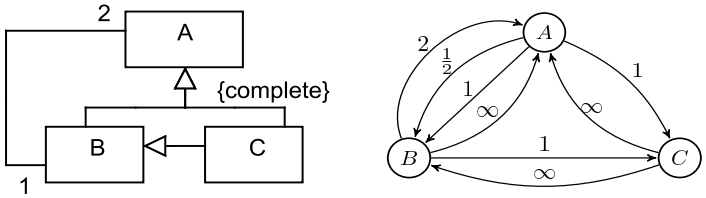


Fig. 19: A non-finitely satisfiable class diagram whose identification graph does not include a critical cycle or a critical set of GS paths

— ***Const*** = $\langle disjoint \rangle$ ***or*** $\langle disjoint, complete \rangle$: $\sum_{j=1}^{n} \frac{1}{weight(\pi_{i_j})} > 1$.

— ***Const*** = $\langle disjoint, incomplete \rangle$: $\sum_{j=1}^{n} \frac{1}{weight(\pi_{i_j})} \geq 1$.

DEFINITION 5.6. ***Const*** = $\langle complete \rangle$ ***or*** $\langle overlapping, complete \rangle$:
For a GS ***constraint*** $G = GS(C,\ C_1,\ \ldots,\ C_n; const)$, ***a set of*** n ***to-sub*** G ***paths*** $\{\pi_i\}_{i=1}^{n}$, ***one for each subclass*** C_i, ***is*** critical ***if***

— ***Const*** = $\langle complete \rangle$: $\sum_{i=1}^{n} weight(\pi_i) < 1$.

— ***Const*** = $\langle overlapping, complete \rangle$: $\sum_{i=1}^{n} weight(\pi_i) \leq 1$.

DEFINITION 5.7. ***Const*** = $\langle incomplete \rangle$ ***or*** $\langle overlapping, incomplete \rangle$:
For a GS ***constraint*** $G = GS(C,\ C_1,\ \ldots,\ C_n; const)$, ***a to-super*** G ***path*** π_i ***is*** critical ***if*** $weight(\pi_i) < 1$. ***For the sake of uniform handling, the term "critical set of*** GS ***paths" is used below also in reference to a critical path.***

The following claim shows that critical sets of GS paths are sub-graphs that identify finite satisfiability problems. For a GS G and Π, a set of G paths, $CD_{/G,\Pi}$ denotes the sub-diagram of CD that includes G and the CD constraints that correspond to the edges in Π.

CLAIM 5.8. ***Let*** CD ***be a class diagram,*** $G = GS(C, C_1,\ ..,\ C_n; const)$ ***a*** GS ***constraint in*** CD, ***and*** Π ***a set of*** G ***paths in*** $graph(CD)$. ***If*** Π ***is a critical set of*** G ***paths then*** $CD_{/G,\Pi}$ ***is not finitely satisfiable.***

PROOF. In the Appendix. □

Example 5.9. For the class diagram in Figure 18-a, $\{\langle B,\ A \rangle,\ \langle C,\ A \rangle\}$ is a critical set of GS paths since $\frac{1}{weight(\langle B,\ A \rangle)} + \frac{1}{weight(\langle C,\ A \rangle)} \geq 1$.
In Figure 1, the ***DNASegment*** GS has a $\langle disjoint, complete \rangle$ constraint. The identification graph in Figure 17 includes the critical set of GS paths
$\{\langle Genome,\ DNASegment \rangle,\ \langle Gene,\ DNASegment \rangle\}$:
$\frac{1}{weight(\langle Genome,\ DNASegment \rangle)} + \frac{1}{weight(\langle Gene,\ DNASegment \rangle)} = 1 + \epsilon > 1$, for $\epsilon > 0$.

Critical sets of GS paths do not provide complete identification of finite satisfiability problems. Figure 19-a (same as Figure 12-a in Section 3) shows a class diagram that is not finitely satisfiable, but its identification graph in Figure 19-b includes neither a critical cycle nor a critical set of GS paths. Moreover, this figure includes a single GS constraint, implying that the *Only-if* direction of claim 5.8 does not hold.

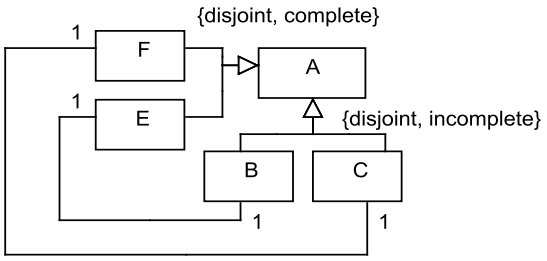


Fig. 20: A finite satisfiability problem that is not identified by a critical cycle or a critical set of GS paths

Figure 20 presents a finite satisfiability problem caused by interaction of GS constraints, that is detected by ***FiniteSat*** but is not identified by a critical cycle or a critical set of GS paths[14]. It seems that the identification graph data structure is too weak for supporting finite satisfiability identification in presence of GS constraints. A stronger data structure that reflects GS constraints is needed.

The patterns catalog in [BGU Modeling Group 2010b] includes multiple patterns for identification of finite satisfiability problems. Many patterns are justified by the results of identification by critical cycles and critical sets of GS paths. Recently, we have checked the educational value of the correctness patterns, and found that previous exposure to the patterns helps designers in detecting and identifying finite satisfiability problems in their designs [Maraee et al. 2011].

6. CONCLUSION AND FUTURE WORK

This paper deals with the problem of finite satisfiability of class diagrams, a problem that is central to the correctness of class diagrams, and relevant in industrial domains such as configuration management [Falkner et al. 2010]. The paper introduces the ***FiniteSat*** algorithm, an efficient method for detecting finite satisfiability problems in class diagrams with multiplicity constraints, class hierarchy and generalization set constraints. ***FiniteSat*** scope is determined by the *structure of class hierarchy* constraints, a new novel parameter, not considered earlier elsewhere.

The ***FiniteSat*** algorithm is strengthened by pre-processing algorithms that propagate implicit *disjoint* and *incomplete* constraints, prior to the application of ***FiniteSat***. In any case, the efficient ***FiniteSat*** cannot apply to all UML class diagrams since deciding finite satisfiability is EXPTIME-complete. For class diagrams that include class hierarchy cycles with *disjoint* or *complete* constraints, we suggest considering relaxation, for example, by removing superclasses.

Detection of finite satisfiability is followed by identification of constraints that cause finite satisfiability problems. Our intention is to link cause identification with the catalog of correctness patterns [BGU Modeling Group 2010b], so to produce human understandable explanations, repair advices, and offer automatic application of repairs. Moreover, the propagation algorithms can help developers by highlighting implicit constraints. In particular, finite satisfiability problems that are caused by *overlapping* and/or *incomplete* constraints point to their redundancy, when considering the optional interpretation of these constraints in UML.

[14]A similar structure occurs in the Electronic-interlocking-system example in the FiniteSatUSE site [BGU Modeling Group 2011a]. The structure, which is a typical pattern of parallel class hierarchies, is marked in the explanation for version 2. The class diagram is not finitely satisfiable, but the problem is not identified by a critical cycle or a critical set of GS paths.

The methods presented in this paper are implemented in an inter-related manner in the ***FiniteSatUSE*** tool [BGU Modeling Group 2011b]. The plan is to add this tool, which is currently embedded within the USE environment, as a plugin of USE. The ***FiniteSat*** algorithm has been extended and applies to additional class diagram constraints, including association class, qualifier, subsetting, redefinition and XOR constraints. This extension is part of a wider analysis of inter-association class diagram constraints [Maraee and Balaban 2011]. The extension to association class enables reduction of finite satisfiability decision to meaningful description logics [Balaban and Maraee 2008]. The ***FiniteSatUSE*** tool is also integrated into our Eclipse based model level integrated development environment [BGU Modeling Group 2011c]. In the future, we plan to strengthen the cause identification methods. Another planned extension involves instance generation and completion, following the constructive proof of ***FiniteSat*** correctness.

This work has several additional results. First, the ***FiniteSat*** algorithm shows that for UML class diagrams as above, but without class hierarchy cycles that include *disjoint* or *complete* constraints, finite satisfiability can be decided in polynomial time (the problem is EXPTIME-complete for UML class diagrams).

Another interesting issue involves the inter-association constraints used in the correctness proof of the ***FiniteSat*** algorithm. The intermediate construction creates a class diagram with *disjoint, overlapping, complete, incomplete* constraints on sets of associations that have a base class, e.g., the XOR constraint. The proof shows that such constraints are interesting and relevant, and provides hints for proving finite satisfiability in presence of such constraints.

Appendix

Proofs for Section 3

Claim 3.4 *If a non-empty finite legal instance I' of CD' satisfies the single class property then $\Gamma'(I')$ is a non-empty finite legal instance of CD.*

PROOF. $\Gamma'(I')$ is a non-empty and finite instance of CD, since I' is non-empty and finite (based on the definition of Γ'). We show that $\Gamma'(I')$ satisfies the constraints of CD.

(1) **Multiplicity constraints**: CD' has all associations and multiplicity constraints of CD. Therefore, for an association a of CD, $a^{I'}$ satisfies the multiplicity constraints on a. The single class property guarantees that ISA related objects that are collapsed under Γ' into a single object, belong to different classes. Consider $c' \in C^{I'}$, where C is a class associated by a. By definition of Γ', $c = \Gamma'(c') \in C^{\Gamma'(I')}$. All a links of c' are mapped under Γ' into a links on c. Due to the single class property, Γ' does not unify different a links of c' (because the links are to objects of the same class), and does not add new a links to c (because if c' is collapsed into a new object, then the other collapsed objects are not from $C^{I'}$).

(2) **Class hierarchy constraints**: A class hierarchy constraint $C \prec D$ in CD is translated in CD' into an ISA association between C and D, with a 1..1 participation constraint for C. Therefore, an object $c \in C^{I'}$ is ISA related to a single object $d \in D^{I'}$, and they are collapsed by Γ' into a single new object o, such that $o \in C^{\Gamma'(I')}$ and $o \in D^{\Gamma'(I')}$. Therefore $C^{\Gamma'(I')} \subseteq D^{\Gamma'(I')}$.

(3) GS **constraints**: Let $GS(C, C_1, \ldots, C_n;\ const)$ be a GS in CD. Its CD' translation is into n ISA associations between C to the n sun-classes, and a constraint on these ISA associations. An ISA linked object structure in I', that instantiates these ISA

associations, satisfies this constraint, and is collapsed into a single object $o \in C^{\Gamma'(I')}$ in $\Gamma'(I')$.
If *const* = *disjoint*, then the object structure instantiates a single ISA link, and therefore in $\Gamma'(I')$ o belongs to a single subclass extension. If *const* = *overlapping*, then there is an object structure in I' that instantiates at least two ISA associations, and therefore in $\Gamma'(I')$ o belongs to at least two subclasses. If *const* = *complete*, then every object in $C^{I'}$ is ISA linked to an object in a subclass, and therefore belongs to one of the subclasses in $\Gamma'(I')$. If *const* = *incomplete*, then there is an object in $c \in C^{I'}$ that is not ISA related to any object of the subclasses. Therefore, $\Gamma'(c)$ belongs to $C^{\Gamma'(I')}$, and does not belong to any subclass extension in $\Gamma'(I')$.

□

Claim 3.5 *Single class property existence*

(*1*) *If CD has a tree or acyclic class hierarchy structure, then every legal instance of CD' satisfies the single class property.*
(*2*) *If the class hierarchy structure of CD does not include cycles with a* disjoint *or a* complete *constraint, then a finitely satisfiable CD' has a non-empty finite legal instance I'' that satisfies the single class property. Moreover, I'' can be constructed from any non-empty finite legal instance, by a procedure that preserves class and association sizes.*

The proof of part (2) depends on a procedure for constructing a non-empty finite legal instance of CD' that satisfies the single class property. For the sake of clarity we separate the procedure from the proof. The construction follows the line of the constructive proof in [Lenzerini and Nobili 1990].
Procedure *Instance Construction*
Input: I', a non-empty finite legal instance of CD', in which the class hierarchy structure does not include cycles with a *disjoint* or a *complete* constraint.
Output: I'', a non-empty finite legal instance of CD' that satisfies the single class property, and preserves the sizes of classes and associations.
Method:

(1) For each class C in CD' insert $|C^{I'}|$ new objects to $C^{I''}$ and number them arbitrarily, starting from 0. For each association a in CD' prepare $|a^{I'}|$ slots and number them, starting from 0.
(2) For every GS constraint $GS\{C, C_1, \ldots, C_n; const\}$ such that $const$ includes *disjoint* or *complete* (i.e., $const \in \{\{$*disjoint*$\}, \{$*complete*$\}, \{$*disjoint, complete*$\}, \{$*disjoint, incomplete*$\}, \{$*overlapping, complete*$\}\}$):
For $i = 1..n$, ranging over the ISA associations $isa_1, \ldots, isa_n$ in this GS constraint, do:
   (a) Reorder $C^{I''}$ such that objects that are not connected by some $isa_j,\ j = 1..n$ to an object in C_j appear in the beginning, and renumber $C^{I''}$, starting from 0.
   (b) Apply to isa_i step (3) below.
(3) For each association $a(rn_1 : C_1[min_1, max_1], rn_2 : C_2[min_2, max_2])$ not populated in step (2) above:
   (a) For $i = 1..2$ do:
      i. For $j = 0$ to $|a^{I'}| - 1$ do: Let $k = j \ mod\ |{C_i}^{I'}|$. Connect the k-th object of ${C_i}^{I''}$, c_{ik} to the j-th slot of a, a_j.
      ii. Renumber a slots so that instances that are connected to the same class objects are numbered contiguously.

(b) $a^{I''}$ is the set of all pairs in the slots of a.

Example A.1. Assume that CD' includes a single association $a(rn_1 : C_1[1,2],\ rn_2 : C_2[1,3])$. Let I' be a non-empty finite legal instance of CD', in which $|C_1^{I'}| = 2,\ |C_2^{I'}| = 3,\ |a^{I'}| = 4$.

Step 1:. $C_1^{I''} = \{c_{1,0},\ c_{1,1}\},\ C_2^{I''} = \{c_{2,0},\ c_{2,1},\ c_{2,2}\}$. The slot set of a: $\{a_0,\ a_1,\ a_2,\ a_3\}$.
Step 3.a.i:. i=1 (Class C_1):
$j = 0$: $k = 0\ mod\ 2 = 0$: Connect: $\langle c_{1,0},\ a_0\rangle$.
$j = 1$: $k = 1\ mod\ 2 = 1$: Connect: $\langle c_{1,1},\ a_1\rangle$.
$j = 2$: $k = 2\ mod\ 2 = 0$: Connect: $\langle c_{1,0},\ a_2\rangle$.
$j = 3$: $k = 3\ mod\ 2 = 1$: Connect: $\langle c_{1,1},\ a_3\rangle$.
Step 3.a.ii:. Renumbered connections: $\langle c_{1,0},\ a_0\rangle,\ \langle c_{1,0},\ a_1\rangle,\ \langle c_{1,1},\ a_2\rangle,\ \langle c_{1,1},\ a_3\rangle$
Step 3.a.i: . i=2 (Class C_2):
$j = 0$: $k = 0\ mod\ 3 = 0$: Connect: $\langle c_{2,0},\ a_0\rangle$.
$j = 1$: $k = 1\ mod\ 3 = 1$: Connect: $\langle c_{2,1},\ a_1\rangle$.
$j = 2$: $k = 2\ mod\ 3 = 2$: Connect: $\langle c_{2,2},\ a_2\rangle$.
$j = 3$: $k = 3\ mod\ 3 = 0$: Connect: $\langle c_{2,0},\ a_3\rangle$.
Step 3.a.ii:. No renumbering of connections is needed.
Step 3.b:. $a^{I''} = \{\langle c_{1,0},\ c_{2,0}\rangle,\ \langle c_{1,0},\ c_{2,1}\rangle,\ \langle c_{1,1},\ c_{2,2}\rangle,\ \langle c_{1,1},\ c_{2,0}\rangle\}$.

PROOF. (Claim 3.5):
1. Tree or acyclic class hierarchy structure:

ISA associations in CD' have a 1..1 multiplicity constraint on the superclass end, and a 0..1 multiplicity constraint on the subclass end. Therefore, given a legal instance I' of CD', an object path $a_1, \ldots, a_n$ of ISA linked objects in I', corresponds to a path of ISA related classes $A_1, \ldots, A_n$ in CD', such that $a_1 \in {A_1}^{I'}, \ldots, a_n \in {A_n}^{I'}$ (every link corresponds to a different ISA association). If the object path includes objects from the same class, then for some i, j, $A_i = A_j$, implying an ISA association cycle, which means that the class hierarchy of CD is cyclic.

2. Class hierarchy structure does not include cycles with a *disjoint* or a *complete* constraint:
First we prove for the case where there are no GS constraints, and then prove for the general case of cyclic class hierarchy structure with GS constraints.

Class hierarchy structure without GS constraints:
Let I' be a non-empty finite legal instance of CD', and I'' the instance constructed by procedure *Instance Construction*. We have to prove three properties: (1) I'' is a non-empty finite instance that preserves the sizes of classes and associations; (2) I'' is a legal instance of CD'; (3) I'' satisfies the single class property.
(1) I'' preserves sizes of classes and associations: I'' is a non-empty finite instance of CD' by its construction. Step 1 of procedure *Instance Construction* populates every class C by $C^{I'}$ objects, and prepares $a^{I'}$ slots for every association a. Therefore, the sizes of all classes are preserved. For associations, the sizes are preserved if all slots of an association are connected to different pairs of objects. Consider an association $a(rn_1 : C_1[min_1, max_1], rn_2 : C_2[min_2, max_2])$. After the first execution of the outer loop $(i = 1)$ of procedure *Instance Construction*, the size of the largest contiguous block of a slots that are connected to the same instance from C_1 is $\lceil \frac{|a^{I'}|}{|{C_1}^{I'}|} \rceil$. Since I' is an instance of CD', $|a^{I'}| \leq |{C_1}^{I'}| \cdot |{C_2}^{I'}|$, implying $\lceil \frac{|a^{I'}|}{|{C_1}^{I'}|} \rceil \leq |{C_2}^{I'}|$. The

second execution of the outer loop ($i = 2$) connects the ordered set of a slots to the ordered set $C_2^{I''}$ in repeated rounds of size $|{C_2}^{I'}|$. By the last inequality, the size of every round is not smaller than the size of a blocks, and therefore repetitions of $C_2^{I''}$ connections do not occur within a continuous a block (the largest set of a slots that are connected to the same C_1 and C_2 instances has size $\lceil \frac{|a^{I'}|}{|{C_1}^{I'}| \cdot |{C_2}^{I'}|} \rceil \leq 1$).

(2) I'' is a legal instance: We have to show that I'' satisfies all multiplicity constraints. I' is a legal instance of CD' and I'' preserves the sizes of class and association extensions in I'. Therefore, for every association $a(rn_1 : C_1[min_1, max_1], rn_2 : C_2[min_2, max_2])$ the following inequalities hold: $min_2 \cdot |{C_1}^{I''}| \leq |a^{I''}| \leq max_2 \cdot |{C_1}^{I''}|$ and $min_1 \cdot |{C_2}^{I''}| \leq |a^{I''}| \leq max_1 \cdot |{C_2}^{I''}|$. By its construction, the number of a instances that are linked to an object in $C_i^{I''}$, for $i = 1..2$, is $\lfloor \frac{|a^{I''}|}{|{C_i}^{I''}|} \rfloor$ or $\lceil \frac{|a^{I''}|}{|{C_i}^{I''}|} \rceil$. Based on the multiplicity inequalities: $min_2 \leq \lfloor \frac{|a^{I''}|}{|{C_1}^{I''}|} \rfloor \leq \lceil \frac{|a^{I''}|}{|{C_1}^{I''}|} \rceil \leq max_2$, and $min_1 \leq \lfloor \frac{|a^{I''}|}{|{C_2}^{I''}|} \rfloor \leq \lceil \frac{|a^{I''}|}{|{C_2}^{I''}|} \rceil \leq max_1$. Therefore, I'' satisfies the multiplicity constraints of CD'.

(3) I'' satisfies the single class property: For an ISA association $isa(sub : C_1[0,1], super : C_2[1,1])$, $isa^{I''} = \{\langle c_{1,i}, c_{2,i} \rangle\}_{i=0}^{|{isa}^{I''}|-1}$ (since $|C_1^{I''}| = |isa^{I''}| \leq |C_2^{I''}|$). Class ordering in procedure *Instance Construction* is performed once, prior to association construction. Therefore, all objects in an ISA-linked object structure have the same serial number, implying that class repetition involves a single object. Therefore, I'' satisfies the single class property.

Consequently, the claim holds if there are no GS constraints, for any class hierarchy structure.

Class hierarchy structure without *disjoint* or *complete* constraints in cycles: Let I' be a non-empty finite legal instance of CD', and I'' the instance constructed by procedure *Instance Construction*. Again, we have to prove the three properties: (1) I'' is a non-empty finite instance that preserves the sizes of classes and associations; (2) I'' is a legal instance of CD'; (3) I'' satisfies the single class property.

Property (1) holds for the same reasons as above, since the process of populating associations is not changed. For property (2), all multiplicity constraints hold, as proved above. We have to show that all GS constraints hold:

(2) I'' satisfies all GS constraints: Consider a GS constraint $C, C_1, \ldots, C_n; const$ in CD'.

(1) *Const* = *disjoint*: Since the constraint holds in I', $|C^{I'}| \geq \sum_{i=1}^{n} |C_i^{I'}|$. Therefore, prior to the population of any isa_i association in this GS there are still at least $|C_i^{I'}|$ objects in $C^{I''}$ that are not connected in any of the already populated isa_j-s associations (due to the $1..1$ multiplicity constraint on isa associations). The *Instance Construction* procedure takes care that the yet not connected $C^{I''}$ objects are moved to the beginning of $C^{I''}$ ordering, and therefore every object in $C^{I''}$ is connected in at most a single isa_i association in this GS, implying that this *disjoint* GS constraint holds in I''.

(2) *Const* = *complete*: The *Instance Construction* procedure takes care that prior to the population of any isa_i association in this GS, the yet not connected $C^{I''}$ objects are moved to the beginning of $C^{I''}$ ordering. Therefore, the procedure guarantees that an object in $C^{I''}$ can be repeatedly connected by some isa_i associations only if all

$C^{I''}$ objects are already connected. Since population of an isa_i association connects exactly $C_i^{I''}$ objects from $C^{I''}$, and since $|C^{I'}| \leq \sum_{i=1}^{n} |C_i^{I'}|$ (I' satisfies the *complete* constraint), all $C^{I''}$ are connected by some isa_i, implying that this *complete* GS constraint holds in I''.

(3) *Const* = *incomplete*: The *incomplete* constraint requires that at least one $C^{I''}$ object is not connected by any isa_i association. Therefore in I', $|C^{I'}| > |C_i^{I'}$ for all $1 \leq i \leq n$. Since $C^{I''}$ objects are ordered only once, the population of every isa_i association uses $|C_i^{I''}|$ objects from the beginning of the ordering of $C^{I''}$, and leaves at least one $C^{I''}$ not connected, implying that this *incomplete* GS constraint holds in I''.

(4) *Const* = *overlapping*: The *overlapping* constraint requires existence of a $C^{I''}$ object which is connected by at least two isa_i. Since every isa_i population uses $C^{I''}$ objects from the beginning of the ordering of $C^{I''}$, at least the first object in $C^{I''}$ is related by all isa_i associations.

Satisfaction in I'' of pair GS constraints is proved by combining the arguments for the single constraints.

(3) I'' satisfies the single class property: The proof for the case of unconstrained class hierarchy applies to all GS constraints that do not include *disjoint* or *complete*. For GS constraints that include *disjoint* or *complete*, the claim assumes that they do not occur in class hierarchy cycles. Therefore, every ISA-related instance structure results from an acyclic class hierarchy structure, for which part (1) of this claim shows that every instance has the single class property.

Consequently, the claim holds also for general class hierarchy structure with GS constraints, provided that *disjoint* or *complete* constraints do not occur in (undirected) class hierarchy cycles.

□

Claim 3.8 If CD' is finitely satisfiable, then Ψ^{CD} is solvable.

PROOF. Let I' be a non-empty finite legal instance of CD'. Let v be a variable value assignment that assigns variables the sizes of their corresponding extensions in I': $v(t) = |T^{I'}|$ for a variable t that represents the element T (class or association). There are three kinds of inequalities introduced for multiplicity, class hierarchy and GS constraints.

(1) **Multiplicity constraint inequalities**: Satisfied, as shown in [Lenzerini and Nobili 1990].

(2) **Class hierarchy inequalities**: A constraint $B \prec A$ in CD inserts the inequality $b \leq a$, where a, b are the variables that represent classes A, B, respectively. In CD', this constraint is replaced by an ISA association $isa(sub : B[0,1],\ super : A[1,1])$, and the v value assignment satisfies the following inequalities (as shown in [Lenzerini and Nobili 1990]):

$$b \leq isa \leq b \quad isa \leq a \quad b \leq a$$

These inequalities reduce to the $b \leq a$ inequality, inserted by ***FiniteSat*** .

(3) GS **constraint inequalities**: Consider a constraint $GS(C, C_1, \ldots, C_n; Const)$, where $c, c_1, \ldots, c_n$ are the variables that represent classes $C, C_1, \ldots, C_n$, respectively.

(a) *const* = *disjoint*: The inequality is $c \geq \sum_{i=1}^{n} c_i$. In CD' this GS constraint is replaced by n ISA associations between C to the C_is, and a similar GS constraint,

imposed on these associations. Therefore, in I', each object of ${C^{I'}}$ can be ISA linked to at most one object of the subclasses, and each object in a subclass is linked to exactly one object from ${C^{I'}}$. Henceforth, $|{C^{I'}}| \geq \sum_{i=1}^{n} |{C_i}^{I'}|$, implying that the *disjoint* inequality is satisfied by v.

(b) *const* = *complete*: The inequality is $c \leq \sum_{i=1}^{n} c_i$. The corresponding GS constraint in CD' requires that in I', every ${C^{I'}}$ object is ISA linked to at least one object of a subclass (and maybe more). Since each object in a subclass is linked to exactly one object from ${C^{I'}}$, $|{C^{I'}}| \leq \sum_{i=1}^{n} |{C_i}^{I'}|$. Therefore, the *complete* inequality is satisfied by v.

(c) *const* = *incomplete*: The inequality is $\forall i \in [1,n].c > c_i$. The corresponding GS constraint in CD' requires that ${C^{I'}}$ includes an object that is not ISA linked. Therefore, $\forall i \in [1,n], |{C^{I'}}| > |{C_i}^{I'}|$, implying that the *incomplete* inequality is satisfied by v.

(d) *const* = *overlapping*: There is no inequality.

Equalities of pair constraints are satisfied by v since they combine the equalities of their components.

□

Claim 3.9 If Ψ^{CD} is solvable, then CD' is finitely satisfiable.

PROOF. The variables of Ψ^{CD} are classified into *class* and *association* variables. We assume that variable c represents class C and variable a represents association a.

The proof is constructive: For a given integer solution v, that for every association a between classes C_1 and C_2 satisfies the condition:

$$v(a) \leq v(c_1) \cdot v(c_2), \qquad \text{(association-size)}$$

we construct a non-empty finite legal instance for CD'. Moreover, the sizes of the classes and associations of the constructed instance are defined by the values of their corresponding variables. The construction mimics the proof of part (2) in Claim 3.5. In that proof, a non-empty finite legal instance I'' for CD' is constructed, based on a given non-empty finite legal instance I'. The construction uses the sizes of classes and associations in I' for populating the classes and associations of I''. The proof of the current claim uses the values that the solution assigns to the variables of Ψ^{CD} for populating the classes and associations of the constructed instance. In fact, the current proof can be automatically obtained from the proof of Claim 3.5 by consistently substituting the size of a class or an association in I' by its representing variable.

The analogy (and hence correctness of the current proof) holds since condition *association-size* and the inequalities of Ψ^{CD} capture the necessary constraints imposed in CD'. For example, for an association $a(rn_1 : C_1[min_1, max_1], rn_2 : C_2[min_2, max_2])$, the above proof relies on the inequalities $min_2 \cdot |{C_1}^{I''}| \leq |a^{I''}| \leq max_2 \cdot |{C_1}^{I''}|$, $min_1 \cdot |{C_2}^{I''}| \leq |a^{I''}| \leq max_1 \cdot |{C_2}^{I''}|$, and $|a^{I'}| \leq |C_1^{I'}| \cdot |C_2^{I'}|$, which are satisfied by I'. Indeed, Ψ^{CD} has the corresponding inequalities $min_2 \cdot c_1 \leq a \leq max_2 \cdot c_1$ and $min_1 \cdot c_2 \leq a \leq max_1 \cdot c_2$, and we assume that v satisfies condition *association-size*. For a GS constraint $C, C_1, \ldots C_n; \{disjoint\}$, the above proof relies on the inequal-

ity $|C^{I'}| \geq \sum_{i=1}^{n} |C_i^{I'}|$, which I' satisfies. Indeed, Ψ^{CD} has the corresponding inequality $c \geq \sum_{i=1}^{n} c_i$. This analogy holds for all other GS constraints.

It remains to show that if Ψ^{CD} is solvable, it has an integer solution that satisfies condition *association-size*. Ψ^{CD} is a linear homogeneous inequality system with rational coefficients. In every solution of Ψ^{CD}, the value assignments for class variables are positive, and the value assignments for association variables are non-negative. Therefore, the solutions of Ψ^{CD} are non-negative. By [Korovin and Voronkov 2001; Lichtman 1978; Calvanese and Lenzerini 1994], if Ψ^{CD} has a real-valued solution, then it also has an integer solution. As for condition *association-size*, since value assignments to class variables are positive, if the condition is not satisfied by a given integer solution, then multiplication by a sufficiently large integer produces a solution that satisfies this condition. □

Proofs for Section 4

Proposition 4.4 Let G be a disjoint GS constraint for a superclass A, with a set of subclasses $\mathcal{S}$. If $\mathcal{S}$ is covered by another set in $A^{\prec^* dis}$, G is redundant. That is, removing it from $\overline{CD}$ does not weaken ***FiniteSat*** .

PROOF. We need to show equivalence of the inequality systems. Let G' be the disjoint GS constraint for A, whose set of subclasses $\mathcal{S}'$ covers $\mathcal{S}$ in $A^{\prec^* dis}$. Let $\mathcal{S} = \{D_1, \ldots, D_n\}$ and $\mathcal{S}' = \{D'_1, \ldots, D'_m\}$, and the inequalities for G and G' be $a \geq d_i,\ i = 1..n,\ a \geq \sum_{i=1}^{n} d_i$ and $a \geq d'_i,\ i = 1..m,\ a \geq \sum_{i=1}^{m} d'_i$, respectively.

First we analyze the forms of mappings from $\mathcal{S}$ to $\mathcal{S}'$:

LEMMA A.2. *The $\prec^*$ mappings of D_i-s to D'_j-s take one of the following forms:*

(*1*) *$D_i \in \mathcal{S}$ is mapped to itself in $\mathcal{S}'$.*
(*2*) *$D_i \in \mathcal{S}$ is mapped to some $D'_j \in \mathcal{S}'$, such that $D_i \prec^* D'_j$, and is the only class in $\mathcal{S}$ that is mapped to D'_j.*
(*3*) *There is a set of classes $D_{i_1}, \ldots D_{i_k}$, for $k > 1$ in $\mathcal{S}$, that are mapped to some $D'_j \in \mathcal{S}'$, and $D_{i_l} \prec^* D'_j$ for all $1 \leq l \leq k$.*

This Lemma is proved by simple analysis of the definition of disjoint descendant set covering. Note that the third case implies that if a set of l D_i-s is mapped to some $D'_j \in \mathcal{S}'$, then $D_i \neq D'_j$, for $i = 1..l$.

It is sufficient to show that in all three forms of the mappings from $\mathcal{S}$ to $\mathcal{S}'$, the inequalities that ***FiniteSat*** generates for the G', imply the inequalities that it generates for G. Therefore, the removed inequalities do not affect the set of solutions.

(1) **Self mapping:** If a D_k in $\mathcal{S}$ is mapped to itself in $\mathcal{S}'$, then its class hierarchy inequality for G is the same as for G'.
(2) **A single ancestor mapping:** There exists a D_k in $\mathcal{S}$ such that $D_k \prec^* D'_j$, for some $D'_j \in \mathcal{S}'$, and is the only class in $\mathcal{S}$ that is mapped to D'_j. Since $D_k \prec^* D'_j$, there is a class hierarchy chain $D_k = C_0 \prec C_1 \preceq \ldots \preceq C_l = D'_j$ in CD, for which ***FiniteSat*** creates the inequalities $d'_j = c_l \geq c_{l-1} \geq \ldots \geq c_0 = d_k$, for some $l \geq 0$. Therefore:
    (a) The inequality $a \geq d'_j$ implies the inequality $a \geq d_k$;
    (b) The inequality $a \geq \sum_{i=1}^{m} d'_i$ implies the inequality $a \geq \sum_{i=1}^{j-1} d'_i + d_k + \sum_{i=j+1}^{m} d'_i$.
(3) **A set ancestor mapping:** For some $D'_j \in \mathcal{S}'$, the set of classes $D_{i_1}, \ldots D_{i_k}$ in $\mathcal{S}$, that are mapped to D'_j satisfies: $k > 1$ and $D_{i_l} \prec^* D'_j$ for all $1 \leq l \leq k$.

(a) The class hierarchy inequalities $a \geq d_{i_l}$ are implied, as above;
(b) The set $\{D_{i_1}, \ldots, D_{i_k}\}$ is in $D'^{\prec^* dis}_j$, and therefore, $\overline{CD}$ includes the disjoint GS constraint $GS(D'_j, D_{i_1}, \ldots, D_{i_k}; \{disjoint\})$, for which ***FiniteSat*** creates the inequality $d'_j \geq \sum_{l=1}^{k} d_{i_l}$. Altogether, inequality $a \geq \sum_{i=1}^{m} d'_i$ implies the inequality $a \geq \sum_{i=1}^{j-1} d'_i + \sum_{l=1}^{k} d_{i_l} + \sum_{i=j+1}^{m} d'_i$.

Applying replacements of the last two cases (single ancestor mapping and set ancestor mapping) to all mappings from $\mathcal{S}$ to $\mathcal{S}'$, yields the disjoint inequality that ***FiniteSat*** creates for G (for self mappings no replacement is needed). Therefore, the inequalities that ***FiniteSat*** creates for G are implied from those created for G', implying that G is redundant. □

Proofs for Section 5

Claim 5.2 Let CD be a class diagram and γ a cycle in $graph(CD)$. Then, $CD_{/\gamma}$ is not finitely satisfiable if and only if γ is critical.

PROOF. $graph(CD)$ is a compact form of the identification graph of [Lenzerini and Nobili 1990], which includes also association nodes: For an association $r(rn_1 : A_1[min_1, max_1], \ rn_2 : A_2[min_2, max_2])$, the edges $\langle A_1, r\rangle$ and $\langle r, A_2\rangle$, which are labeled max_2 and $\frac{1}{min_1}$, respectively, are compacted into a single edge $\langle A_1, A_2\rangle$ labeled $\frac{max_2}{min_1}$ (and similarly for the edges from A_2 to A_1 through r).

A critical cycle whose edges correspond to multiplicity constraints is a compact form of a critical cycle in the identification graph of [Lenzerini and Nobili 1990]. Therefore, the claim holds for such cycles, since [Lenzerini and Nobili 1990] shows this claim for their identification graph.

For cycles with edges that correspond to class hierarchy constraints, we notice that the edges constructed in $graph(CD)$ for a class hierarchy constraint $B \prec A$ in CD are the ones that are constructed for the association $isa(sub : B[0, 1], super : A[1, 1])$. In the correctness proof of Algorithm ***FiniteSat*** (Theorem 3.6), it is shown that under the above transformation, that replaces the class hierarchy constraint by the association, finite satisfiability is not affected. Therefore, the claim holds also for cycles with edges that correspond to class hierarchy constraints. □

Theorem 5.4. A class diagram CD with multiplicity and class hierarchy constraints alone is not finitely satisfiable if and only if $graph(CD)$ includes critical cycles.

PROOF. By Claim 5.2, if the class diagram includes a critical cycle then it is not finitely satisfiable. For the opposite direction we note that the transformation that replaces all class hierarchy constraints $B \prec A$ in CD by corresponding associations $isa(sub : B[0, 1], super : A[1, 1])$ in CD', preserves finite satisfiability. CD' includes only multiplicity constraints, and the result of [Lenzerini and Nobili 1990], that refers to their identification graph, applies to it: If a class diagram is not finitely satisfiable, then its identification graph includes a critical cycle.

It remains to show that if the identification graph of [Lenzerini and Nobili 1990] includes a critical cycle then this is also the case for our identification graph. A cycle in the graph of [Lenzerini and Nobili 1990] consists of either pairs of successive edges $\langle A_1, r\rangle$ and $\langle r, A_2\rangle$ that correspond to multiplicity constraints on two different ends in an association $r(rn_1 : A_1[min_1, max_1], \ rn_2 : A_2[min_2, max_2])$ (kind 1), or of successive edges $\langle A_1, r\rangle$ and $\langle r, A_1\rangle$, that correspond to the multiplicity constraints on one side of the association (kind 2). A pair $\langle A_1, r\rangle$ and $\langle r, A_2\rangle$ in the identification graph of [Lenzerini and Nobili 1990] is captured by a single edge $\langle A_1, A_2\rangle$ in $graph(CD)$ with the same weight. Therefore, if the [Lenzerini and Nobili 1990] identification graph includes a critical cycle that consists only from kind (1) edge pairs, then our identi-

fication graph also includes a critical cycle (the compact form of the [Lenzerini and Nobili 1990] critical cycle). Otherwise, if a [Lenzerini and Nobili 1990] critical cycle includes a kind (2) edge pair, then removing this edge pair still leaves the cycle critical (since these edges correspond to the multiplicity constraint $A_1[min_1, max_1]$, and therefore their weight $\frac{max_1}{min_1} \geq 1$). Consequently, if the [Lenzerini and Nobili 1990] graph includes a critical cycle then it includes also a critical cycle that consist only of kind (1) edge pairs, implying that our graph also includes a critical cycle. □

Claim 5.8 Let CD be a class diagram, $G = GS(C, C_1, .., C_n; const)$ a GS constraint in CD, and $\Pi = \{\pi_1, \ldots \pi_n\}$ a set of G paths in $graph(CD)$. If Π is a critical set of G paths then $CD_{/G,\Pi}$ is not finitely satisfiable.

PROOF. Assume, by negation, that $CD_{/G,\Pi}$ has a non-empty finite legal instance I.

(1) *Const* = {*disjoint*} *or* {*disjoint, complete*}: By the generalization of Proposition 5.1 to paths, for every $1 \leq i \leq n$, $\frac{|C_i^I|}{|C^I|} \leq weight(\pi_i) \Rightarrow \frac{|C_i^I|}{|C^I|} \geq \frac{1}{weight(\pi_i)}$. Therefore, $\sum_{i=1}^{n} \frac{|C_i^I|}{|C^I|} \geq \sum_{i=1}^{n} \frac{1}{weight(\pi_i)} > 1$, implying $\sum_{i=1}^{n} |C_i^I| > |C^I|$, in contradiction to the semantics of *const*. Therefore, I is not a legal instance.

(2) *Const* = {*disjoint, incomplete*}: Proof as above. Only, the last inequality is $\sum_{i=1}^{n} \frac{1}{weight(\pi_i)} \geq 1$, implying $\sum_{i=1}^{n} |C_i^I| \geq |C^I|$, in contradiction to the semantics of *const*.

(3) *Const* = {*complete*}: For every $1 \leq i \leq n$, $\frac{|C_i^I|}{|C^I|} \leq weight(\pi_i)$. Therefore, $\sum_{i=1}^{n} \frac{|C_i^I|}{|C^I|} \leq \sum_{i=1}^{n} weight(\pi_i) < 1$, implying $\sum_{i=1}^{n} |C_i^I| < |C^I|$, in contradiction to the semantics of *const*.

(4) *Const* = {*overlapping, complete*}: Proof as above. Only, the last inequality is $\sum_{i=1}^{n} weight(\pi_i) \leq 1$, implying $\sum_{i=1}^{n} |C_i^I| \leq |C^I|$, in contradiction to the semantics of *const*.

(5) *Const* = {*incomplete*} *or* {*overlapping, incomplete*}: $\frac{|C^I|}{|C_i^I|} \leq weight(\pi_i) < 1$, implying $|C^I| \leq |C_i^I|$, in contradiction to the semantics of *const*.

□

ACKNOWLEDGMENTS

We would like to thank Victor Makarenkov, Rami Prilutsky, Vitaly Bichov and Michael Brichko for their help in constructing the ***FiniteSatUSE*** tool. We thank Gera Weiss and Arnon Sturm for fruitful consultations, and the anonymous referees for their invaluable comments.

REFERENCES

ALANEN, M. AND PORRES, I. 2008. A Metamodeling Language Supporting Subset and Union Properties. *Software and Systems Modeling 7,* 1, 103–124.

ANASTASAKIS, K., BORDBAR, B., GEORG, G., AND RAY, I. 2010. On Challenges of Model Transformation from UML to Alloy. *Software and Systems Modeling 9,* 1, 69–86.

ANDRE, P., ROMANCZUK-REQUILE, A., ROYER, J.-C., AND VASCONCELOS. 2000. Checking the Consistency of UML Class Diagrams Using Larch Prover. In *The*

third Rigorous Object-Oriented Methods Workshop. Electronic Workshops in Computing (eWiC). BCS, The Chartered Institute for IT.

ARTALE, A., CALVANESE, D., AND IBANEZ-GARCIA, A. 2010. Full Satisfiability of UML Class Diagrams. In *Proc. of the 29th Int. Conf. on Conceptual Modeling (ER 2010)*. LNCS Series, vol. 6412. Springer, 317–331.

ARTALE, A., CALVANESE, D., KONTCHAKOV, R., RYZHIKOV, V., AND ZAKHARYASCHEV, M. 2007. Complexity of Reasoning in Entity Relationship Models. In *Proc. of the 2007 Description Logic Workshop (DL 2007)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/` Series, vol. 250. Bozen-Bolzano University Press, Brixen-Bressanone, Italy, 163–170.

BALABAN, M. AND MARAEE, A. 2008. A UML-Based Method for Deciding Finite Satisfiability in Description Logics. In *21st International Workshop on Description Logics*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/` Series, vol. 353. Dresden, Germany.

BALABAN, M., MARAEE, A., AND STURM, A. 2010. Management of Correctness Problems in UML Class Diagrams – Towards a Pattern-Based Approach. *International Journal of Information System Modeling and Design. 1,* 4, 24–47.

BALABAN, M. AND SHOVAL, P. 2002. MEER – An EER Model Enhanced with Structure Methods. *Information Systems 27,* 4, 245–275.

BAYLEY, C. 2004. Modelling Interlocking Systems with UML. In *The IEE Seminar on Railway System Modelling- Not Just for Fun*. 0–3.

BERARDI, D., CALVANESE, D., AND GIACOMO, D. 2005. Reasoning on UML Class Diagrams. *Artificial Intelligence 168*, 70–118.

BERRABAH, D. AND BOUFARÈS, F. 2008. Constraints Satisfaction Problems in Data Modeling. In *Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology*. CSTST '08. ACM, Cergy-Pontoise, France, 292–297.

BGU MODELING GROUP. 2010a. `http://www.cs.bgu.ac.il/~modeling/`.

BGU MODELING GROUP. 2010b. UML Class Diagram Pattern Catalog. `http://www.cs.bgu.ac.il/~cd-patterns/`.

BGU MODELING GROUP. 2011a. Description of the FiniteSatUSE Tool. `http://www.cs.bgu.ac.il/~modeling/?page_id=314`.

BGU MODELING GROUP. 2011b. FiniteSatUSE – A Class Diagram Correctness Tool. `http://sourceforge.net/projects/usefsverif/`.

BGU MODELING GROUP. 2011c. Model-Driven Integrated Development Environment (MIDE). `http://www.cs.bgu.ac.il/~modeling/?page_id=235`.

BLAHA, M. AND PREMERLANI, W. 1997. *Object-Oriented Modeling and Design for Database Applications* 1 Ed. Prentice Hall.

BOUFARES, F. AND BENNACEUR, H. 2004. Consistency Problems in ER-schemas for Database Systems. *Information Sciences 163,* 4, 263–274.

BRUCKER, A. AND WOLFF, B. 2006. The HOL-OCL Book. Tech. Rep. 525, Information Security, Swiss Federal Institute of Technolgy (ETH), 8092 Zurich, Switzerland. August.

BRUCKER, A. AND WOLFF, B. 2008. HOL-OCL: A Formal Proof Environment for UML/OCL. In *Fundamental approaches to software engineering*. LNCS Series, vol. 4961. Springer-Verlag, 97–100.

CABOT, J., CLARISÓ, R., AND RIERA, D. 2007. UMLtoCSP: A Tool for the Formal Verification of UML/OCL Models using Constraint Programming. In *Proceedings of the twenty-second IEEE-ACM international conference on Automated software engineering*. ASE '07. ACM, New York, NY, USA, 547–548.

CABOT, J., CLARISÓ, R., AND RIERA, D. 2008. Verification of UML/OCL Class Diagrams Using Constraint Programming. In *Proceedings of the 2008 IEEE In-*

ternational Conference on Software Testing Verification and Validation Workshop. ICSTW'08. IEEE Computer Society, 73–80.

CADOLI, M., CALVANESE, D., DE GIACOMO, G., AND MANCINI, T. 2004. Finite Satisfiability of UML Class Diagrams by Constraint Programming. In *Proc. of the CP 2004 Workshop on CSP Techniques with Immediate Application*.

CALVANESE, D. AND LENZERINI, M. 1994. On the Interaction Between ISA and Cardinality Constraints. In *Proceedings of the Tenth International Conference on Data Engineering*. IEEE Computer Society, 204 –213.

CHANDA, J., KANJILAL, A., AND SENGUPTA, S. 2010. UML-Compiler: A Framework for Syntactic and Semantic Verification of UML Diagrams. In *Distributed Computing and Internet Technology*. LNCS Series, vol. 5966. Springer Berlin / Heidelberg, 194–205.

CHEN, P. 1976. The Entity-Relationship Model Toward a Unified View of Data. *ACM Transactions on Database Systems (TODS) 1,* 1, 9–36.

ENGEL, K. AND HARTMAN, S. 1995. Constructing Realizers of Semantic Entity Relationship Schemes. Tech. rep., Universitat Rostock, Fachbereich Mathematik, Rostock, Germany.

EUROPEAN RAIL TRAFFIC MANAGEMENT SYSTEM. 2007. Euro- Interlocking: European Standards for Railways Interlocking Systems. `http://ertms.uic.asso.fr/documents/interlocking/interlock_brochure.pdf`.

FALKNER, A., FEINERER, I., SALZER, G., AND SCHENNER, G. 2010. Computing Product Configurations via UML and Integer Linear Programming. *International Journal of Mass Customisation 3,* 4, 351 – 367.

FEINERER, I. 2007. A Formal Treatment of UML Class Diagrams as an Efficient Method for Configuration Management. Ph.D. thesis, Theory and Logic Group, Institute of Computer Languages, Vienna University of Technology.

FEINERER, I., SALZER, G., AND SISEL, T. 2011. Reducing Multiplicities in Class Diagrams. In *Model Driven Engineering Languages and Systems*, J. Whittle, T. Clark, and T. Kühne, Eds. LNCS Series, vol. 6981. Springer-Verlag, 379–393.

FELFERNIG, A., FRIEDRICH, G., AND JANNACH, D. 2001. Conceptual modeling for configuration of mass-customizable products. *Artificial Intelligence in Engineering 15,* 2, 165 – 176.

FLEISHANDERL, G., FRIEDRICH, G., HASELBOCK, A., SCHREINER, H., AND STUMPTNER, M. 1998. Configuring Large Systems Using Generative Constraint Satisfaction. *Intelligent Systems and their Applications, IEEE 13,* 4, 59 –68.

FORMICA, A. 2002. Finite Satisfiability of Integrity Constraints in Object-Oriented Database Schemas. *IEEE Transactions on Knowledge and Data Engineering 14,* 1, 123–139.

GEORG, G., BIEMAN, J., AND FRANCE, R. 2001. Using Alloy and UML/OCL to Specify Run-Time Configuration Management: a Case Study. In *Practical UML-Based Rigorous Development Methods– Countering or Integrating the eXtremists*, A. Evans, R. France, A. Moreira, and B. Rumpe, Eds. LNI Series, vol. P-7. German Informatics Society, 128–141.

GOGOLLA, M., BOHLING, J., AND RICHTERS, M. 2005. Validating UML and OCL Models in USE by Automatic Snapshot Generation. *Journal on Software and System Modeling 4*, 386–398.

GOGOLLA, M., KUHLMANN, M., AND HAMANN, L. 2009. Consistency, Independence and Consequences in UML and OCL Models. In *Proceedings of the 3rd International Conference on Tests and Proofs*. LNCS. Springer-Verlag, 90–104.

GOGOLLA, M. AND RICHTERS, M. 2002. Expressing UML Class Diagram Properties with OCL. In *Object Modeling with the OCL*. LNCS Series, vol. 2263. Springer-Verlag, 423–426.

HARTMANN, S. 1995. Graph-Theoretical Methods to Construct Entity-Relationship Databases. In *Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science*. LNCS Series, vol. 1017. Springer-Verlag, 131–145.

HARTMANN, S. 2001. Coping with Inconsistent Constraint Specifications. In *Proceedings of the 20th International Conference on Conceptual Modeling*. LNCS Series, vol. 2224. Springer-Verlag, London, UK, 241–255.

JACKSON, D. 2002. Alloy: A Lightweight Object Modelling Notation. *ACM Transactions on Software Engineering and Methodology (TOSEM) 11,* 2, 256–290.

JACKSON, D. 2006. *Software Abstractions: Logic, Language and Analysis*. The MIT Press.

JACKSON, D. AND RINARD, M. 2004. Software Analysis: A Roadmap. In *Proceedings of the Conference on the Future of Software Engineering*. ACM, 133–145.

JARRAR, M. AND HEYMANS, S. 2008. Towards Pattern-Based Reasoning for Friendly Ontology Debugging. *International Journal on Artificial Intelligence Tools 17,* 4, 607–634.

KANEIWA, K. AND SATOH, K. 2010. On the Complexities of Consistency Checking for Restricted UML Class Diagrams. *Theor. Comput. Sci. 411,* 2, 301–323.

KLEPPE, A., WARMER, J., AND BAST, W. 2003. *MDA Explained: The Model Driven Architecture(TM): Practice and Promise* 1 Ed. Addison-Wesley Professional.

KOROVIN, K. AND VORONKOV, A. 2001. Verifying Orientability of Rewrite Rules using the Knuth-Bendix Order. In *Rewriting Techniques and Applications*. LNCS Series, vol. 2051. Springer-Verlag, 137–153.

KUHLMANN, M., HAMANN, L., AND GOGOLLA, M. 2011. Extensive Validation of OCL Models by Integrating SAT Solving into USE. In *TOOLS EUROPE 2011: Objects, Models, Components, Patterns*. Vol. 6705. Springer, 290–306.

LANGE, C., CHAUDRON, M., AND J., M. 2006. In Practice: UML Software Architecture and Design Description. *IEEE Software 23,* 2, 40 – 46.

LENZERINI, M. AND NOBILI, P. 1990. On the Satisfiability of Dependency Constraints in Entity-Relationship Schemata. *Information Systems 15,* 4, 453–461.

LICHTMAN, M. L. 1978. A Unified Approach for Finding Real and Integer Solutions to Systems of Linear Inequalities. *Applied Mathematics and Computazton 4*, 177–186.

LUTZ, C., SATTLER, U., AND TENDERA, L. 2005. The Complexity of Finite Model Reasoning in Description Logics. *Information and Computation 199*, 132–171.

MAKARENKOV, V., JELNOV, P., MARAEE, A., AND BALABAN, M. 2009. Finite Satisfiability of Class Diagrams: Practical Occurrence and Scalability of the ***FiniteSat*** Algorithm. In *MoDeVVa '09: Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation*. ACM, 1–10.

MAOZ, S., RINGERT, J., AND RUMPE, B. 2011. CD2Alloy: Class Diagrams Analysis Using Alloy Revisited. In *Model Driven Engineering Languages and Systems*, J. Whittle, T. Clark, and T. Kühne, Eds. LNCS Series, vol. 6981. Springer-Verlag, 592–607.

MARAEE, A. 2007. Efficient Methods for Solving Finite Satisfiability Problems in UML Class Diagrams. M.S. thesis, Ben-Gurion University of the Negev.

MARAEE, A. AND BALABAN, M. 2007. Efficient Reasoning About Finite Satisfiability of UML Class Diagrams with Constrained Generalization Sets. In *The 3rd European Conference on Model-Driven Architecture*. LNCS Series, vol. 4530. Springer-Verlag, 17–31.

MARAEE, A. AND BALABAN, M. 2011. *On the Interaction of Inter-Relationship Constraints*. Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVA 2011), MoDELS 2011.

MARAEE, A., BALABAN, M., STURM, A., AND ASHROV, A. 2011. Model Correctness Patterns as an Educational Instrument. In *7th Educators' Symposium, MODELS*

2011: Software Modeling in Education.

MARAEE, A., MAKARENKOV, V., AND BALABAN, B. 2008. Efficient Recognition and Detection of Finite Satisfiability Problems in UML Class Diagrams: Handling Constrained Generalization Sets, Qualifiers and Association Class Constraints. In *The 1st International Workshop on Model Co-Evolution and Consistency Management, MoDELS 2008*.

NIEDERBRUCKER, G. AND SISEL, T. 2011. Clews Website. `http://www.logic.at/clews/index.html`.

Object Management Group 2006. *UML 2.0 Object Constraint Language Specification*. Object Management Group.

OMG. 2007. The UML 2.0 Superstructure Specification. Specification Version 2, Object Management Group.

PATON, N., KHAN, S., HAYES, A., MOUSSOUNI, F., BRASS, A., EILBECK, K., GOBLE, C., HUBBARD, S., AND OLIVER, S. 2000. Conceptual Modelling of Genomic Information. *Bioinformatics 16,* 6, 548–557.

QUERALT, A., RULL, G., TENIENTE, E., FARRÉ, C., AND URPÍ, T. 2010. AuRUS: Automated Reasoning on UML/OCL Schemas. In *Conceptual Modeling ER 2010*. LNCS Series, vol. 6412. Springer-Verlag, 438–444.

QUERALT, A. AND TENIENTE, E. 2006. Reasoning on UML Class Diagrams with OCL Constraints. In *CONCEPTUAL MODELING - ER 2006*. Vol. 4215. Springer-Verlag.

QUERALT, A. AND TENIENTE, E. 2008. Decidable Reasoning in UML Schemas with Constraints. In *Advanced Information Systems Engineering*. LNCS Series, vol. 5074. Springer-Verlag, 354–254.

SCHILD, K. 1991. A Correspondence Theory for Terminological Logics: Preliminary Report. In *IJCAI*. Morgan Kaufmann Publishers Inc., 466–471.

SCHRIJVER, A. 1998. *Theory of Linear and Integer Programming*. John Wiley & Sons Inc.

SHAIKH, A., CLARISÓ, R., WIIL, U., AND MEMON, N. 2010. Verification-Driven Slicing of UML/OCL Models. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 185–194.

SHAIKH, A., WIIL, U., AND MEMON, N. 2011. UOST: UML/OCL Aggressive Slicing Technique for Efficient Verification of Models. In *6th Workshop on System Analysis and Modelling*. LNCS Series, vol. 6598. Springer-Verlag, 173–192.

SOEKEN, M., WILLE, R., KUHLMANN, M., GOGOLLA, M., AND DRECHSLER, R. 2010. Verifying UML/OCL Models Using Boolean Satisfiability. In *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE '10. European Design and Automation Association, 1341–1344.

SPACCAPIETRA, E., Ed. 1987. *Entity-Relationship Approach: Ten Years of Experience in Information Modeling : Proceedings of the Fifth International Conference on Entity-Relation*. Elsevier Science Ltd.

SUNYE, G., POLLET, D., LE TARAON, Y., AND J.-M, J. 2001. Refactoring UML Models. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, UML 2001*. LNCS Series, vol. 2185. Springer-Verlag, 134–148.

THALHEIM, B. 1992. Fundamentals of Cardinality Constraints. In *The 11th International Conference on the Entity-Relationship Approach*. Springer-Verlag, 7–23.

THALHEIM, B. 2000. *Entity Relationship Modeling, Foundation of Database Technology*. Springer-Verlag.

WAHLER, M., BASIN, D., D. BRUCKER, D., AND KOEHLER, K. 2010. Efficient Analysis of Pattern-Based Constraint Specifications. *Software and Systems Modeling 9,* 2, 225–255.

WARMER, J. AND KLEPPE, A. 2003. *The Object Constraint Language: Getting Your Models Ready for MDA* 2 Ed. Addison-Wesley Longman Publishing Co., Inc.

Received ; revised ; accepted