Ben-Gurion University of the Negev

Faculty of Engineering Sciences

Department of Information Systems Engineering

Efficient Methods for Solving Finite Satisfiability Problems in UML Class Diagrams

by Azzam Maraee

Supervised by: Prof. Mira Balaban

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR M.Sc DEGREE

December 2006

Abstract

This thesis offers methods for detecting and addressing problems of finite satisfiability in UML class diagrams in a way that is simple and efficient and that provides the foundations for expanding UML case tools to address these finite satisfiability problems.

UML class diagrams are the most important and well-established of UML models. UML class diagrams allow the specification of constraints such as cardinality constraints, class hierarchy constraints and inter-association constraints. Constraints extend the expressivity of class diagrams, but enable the specification of unsatisfiable class diagrams, i.e., class diagrams that have a finite and non-empty instance world.

The central role that class diagrams play in the design and specification of software, databases and ontologies highlights the need for powerful CASE tools, at the level of current Integrated Development Environments (*IDE*)s. Such tools need to prevent syntax errors, detect redundancies and contradictions, identify the reasons for errors and possibly suggest design improvements.

This work classifies the existing finite satisfiability problems in UML class diagram with class hierarchy and generalization set constraints. It presents a set of finite reasoning methods for addressing these problems. These methods are based on the transformation of these problems into a set of linear inequalities whose size is polynomial in the size of the diagram. An experimental tool that was implemented is also presented.

Acknowledgements

It is difficult to overstate my gratitude to my supervisor, Professor Mira Balaban. Professor Balaban inspired me to take on this fascinating research topic. Her constant support, enthusiasm, inspiration and great efforts to explain things clearly and simply, made this thesis possible. Her guidance, insights and numerous suggestions were crucial for the successful conclusion of this research project. Throughout my thesis-writing period, Professor Balaban provided encouragement, sound advice, good teaching, good company and lots of good ideas and for that I will be forever indebted to her.

I would like also to extend my gratitude to Dr. Arnon Strom. His many helpful comments, suggestions and insights contributed greatly to the development of this research.

My thanks go to Lior Limonad for hosting me during the Ngits06 conference, in which preliminary results of my research were presented. I am also indebted to Mr. Juergen Wuest for permitting me to use his SDmatric program, which was essential for developing the implementation part of this research.

I am deeply grateful to my brother, Dr. Durgham Maraee. His advice and comments provided constant encouragement and motivated me to bring this research to a successful conclusion. Lastly, and most importantly, I wish to thank my parents and my family for their love and constant support and for believing in me.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Unified Modeling Language (UML) is nowadays the industry standard modeling framework, including multiple visual modeling languages, referred to as UML models. Traditionally, UML models are used for analysis and design of complex systems. Their relevance has increased with the advent of the Model-Driven Development (MDD) approach, in which analysis and design models play an essential role in the process of software development. Recently, with the emergence of web-enabled agent technology, UML models are used also for ontology representation, and construction and extraction of ontologies [6, 7, 5].

In view of their wide popularity, it is highly important that UML models provide reliable support for the designed systems, and be subject to stringent quality assurance and quality control criteria [30]. Indeed, an extensive amount of research efforts is devoted to formalization of UML models, specification of their semantics, and development of reasoning and correctness checking methods [4, 10]. Moreover, with the prevalence of the Model Driven Engineering approach, it is expected that all information in a design model will be effective in its successive models.

Modeling problems usually arise when models are scaled to model large, distributed applications. A model may originate from different sources and a large number of designers can be involved in the modeling process. Designers are highly prone to making mistakes,

and combining information from different sources gives rise to potential conflicts [3, 35, 16]. [20] shows that defects often remain undetected, even if the model is read attentively by practitioners.

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

Class Diagrams are probably the most important and best understood among all UML models. A Class Diagram provides a static description of system components. It describes systems structure in terms of classes, associations, and constraints imposed on classes and their inter-relationships. Constraints provide an essential means of knowledge engineering, since they extend the expressivity of diagrams. UML supports class diagram constraints such as cardinality constraints, class hierarchy constraints, and inter-association constraints. Example 1.1 below, presents a class diagram that includes cardinality and hierarchy constraints.

Example 1.1. *Figure 1.1 presents a class diagram with three classes named Academic, Graduate and FacultyMember, one association advisor-student between instances of the Academic and the Graduate classes, with roles named advisor and student, respectively, a cardinality constraint that is imposed on this association, and a generalization set with a superclass Academic and sub-classes Graduate and FacultyMember. The cardinality constraint states that every Graduate student must be advised by exactly one Academic, while every Academic must advise exactly two Graduate students. The generalization set states that Graduates and FacultyMembers are Academic as well, implying that the advisor of a Graduate can be a Graduate or a FacultyMember or another Academic.*

In the presence of constraints a class diagram may turn inconsistent, as it might impose constraints that cannot be finitely satisfied. Figure 1.1, presents a multiplicity constraint cycle that involves a compound class, *Graduate*, whose instances must be related to *Academic*

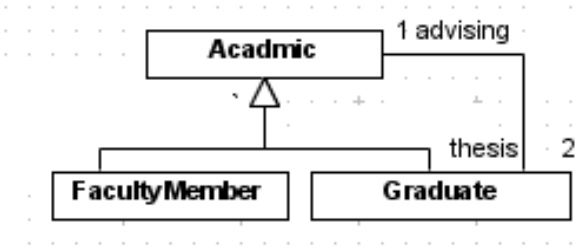


Figure 1.1: A Class Diagram with a Finite Satisfiability Problem

instances. Therefore, the number of *student-advisor* links in every diagram instance must be both, $G \cdot 1$ and $A \cdot 2$, assuming that G and A are the number of graduates and academics, respectively. Therefore, the extensions of *Graduate* and *Academic* must satisfy $G = A \cdot 2$, while the *Graduate* extension is a subset of the *Academic* extension. This constraint can be satisfied only by empty or infinite extensions. Such problems are termed *finite satisfiability* problems.

The problem of finite satisfiability has been studied in the context of various kinds of conceptual schemata [1, 3, 8, 11, 14, 18, 24]. There are methods for testing finite satisfiability, for detecting causes for unsatisfiability, and for heuristic suggestions for diagram correction. Yet, no method provides a feasible solution for detecting lack of finite satisfiability for the combination of cardinality constraints, class hierarchy constraints, and generalization sets constraints.

In this thesis, we present a linear programming based methods for reasoning about finite satisfiability of UML class diagrams that includes class hierarchy constraints, and generalization sets constraints. The methods is based on a reduction to the algorithm of Lenzerini and Nobili [18] that was applied only to ER-diagrams without class hierarchies. It is simple and feasible since it adds in the worst case only a linear amount of entities to the original diagram. It improves over previous extensions of the Lenzerini and Nobili method that require the addition of an exponential number of new entities to the original diagram [8]. An implementation of our method within a UML case tool is presented in Chapter 5.

The thesis is organized as follows: Chapter 2 presents the finite satisfiability notion,

summarizes relevant methods for detecting finite satisfiability problems in class diagrams, introduces the Generalization Set notion of UML2.0, and classifies different class hierarchy structures. Although this thesis focuses only on finite satisfiability problems, for the sake of completeness we also introduce the consistency notion. In Chapter 3 we present polynomial time algorithms for testing satisfiability of UML class diagrams with unconstrained generalization sets. Chapter extends the algorithms to operate on constrained generalization set, and investigate the limits of these methods. Chapters 5 presents our experimental tool. Chapter 6 concludes and draws the line for future research.

Chapter 2

Background

2.1 UML Diagrams

The Unified Modeling Language UML is now the standard graphical modeling language developed and adopted by the Object Management Group for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems [22]. UML simplifies the complex process of software design by raising the level of abstraction throughout the analysis and design process. Their relevance has increased with the advent of the Model Driven Development (MDD) approach, in which analysis and model design play an essential role in the process of software development. Recently, with the emergence of web-enabled agent technology, UML models are used also for ontology representation, construction and extraction.

A central assumption that underlined the development of UML was the idea that it is not possible to describe a complex system with a single model only. A "rich" description of a system must include a number of highly detailed models. UML consists of twelve diagrams referred to as UML models. Table 2.1 summarizes the UML diagrams and the modeling view of software solutions represented by them (extracted from [30]).

UML Diagrams	Represent
Use case	functionality from the user's viewpoint
Activity	the flow within a Use case or the system
Class	classes, entities, business domain, database
Interaction overview Interaction overview	interactions at a general high level
Communication	interactions between objects
Object	objects and their links
State machine	the run-time life cycle of an object
Composite structure	component or object behavior at run-time
Component	executables, link able libraries, etc.
Deployment	hardware nodes and processors
Package	subsystems, organizational units
Timing	time concept during object interactions

Table 2.1: UML Diagrams

2.2 Class Diagram

Among the twelve visual UML models, class Diagrams are probably the most important and best understood among all UML models. UML class diagrams are used to specify, visualize, and document the system static view. They also serve as a basis for generating implementation artifacts such as code skeleton and database schemata, as a means for knowledge representation such as specifying ontologies, and for defining meta-models of other programming, modeling, and specification languages. The origin of the class diagram model is the conceptual models of the 80's, like Entity-relationship (ER) diagrams [53], their Enhanced versions (EER), Object-Role Modeling (ORM) diagrams [54], and Frames structured modeling in artificial intelligence [55]. The UML class diagram model includes elements from all these models.

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 constraints on associations* (also termed cardinality constraint), with or without *qualifiers*; (2) *association*

class constraint; (3) *class* and *associations hierarchy constraints*; (4) *generalization set constraints*; (5) *association constraints*; (6) *aggregation constraints*; (7) *multiplicity constraints on attributes*. The syntax and informal semantics are described in Rumbaugh et al [23] and in OMG-UML [22].

Figure 2.1 is an example of a class diagram, which partially specifies a university system. It captures the people hierarchy within the university and their relationship to the university courses. Classes are represented by rectangles; associations are represented by lines between the rectangles; the qualifiers are presented by small rectangles attached to the end of an association paths; n-ary association is an association among three or more classes, it is shown as a large diamond, with a path from the diamond to each participant class; multiplicity constraints are marked on the association's line ends; association classes are marked by a dashed line connecting a class rectangle with an association line; class hierarchy constraints are marked by empty arrow heads; association hierarchies are marked by a dashed arrow labeled "subset" between association lines; aggregations is a special form of binary association. It presented by a hollow diamond adornment on the end of an association line at which it connects to the aggregate class. If the aggregation is a composition, then the diamond is filled.

The standard set theoretic semantics of class diagrams associates a class diagram with *class diagram instances* in which classes have extensions that are sets of objects that share structure and operations, and associations have extensions that are relationships among class extensions. We denote class diagrams as CD, class symbols as C, association symbols as A, role symbols as rn and instance symbols as I. The application of the meaning of assignment to a symbol T of CD is denoted T^I. Henceforth, we shorten expressions like "instance of an extension of C" by "instance of C" and "instance of an extension of A" by "instance of A". For example, in Figure 1, the *Academic* class represents the set of academic people in the university, the binary association between *FacultyMember* and *Course* denotes a set of pairs of *FacultyMember*s and *Course*s in which the *FacultyMember* plays the role

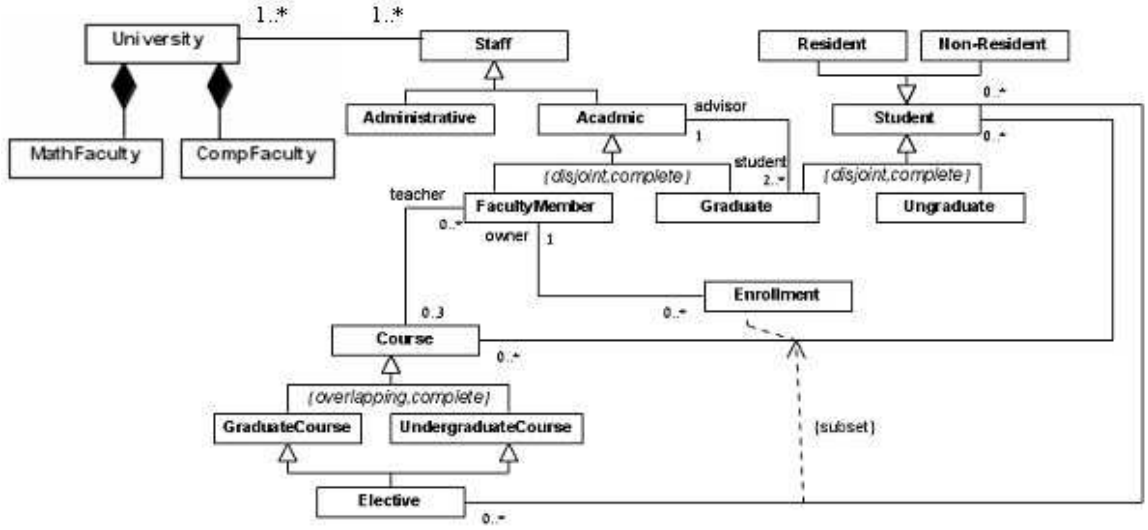


Figure 2.1: UML Class Diagram

of a teacher. The ternary association between *FacultyMember*, *Course* and *Student* denotes a 3-tuple of values, one from each of the respective classes. Attributes serve to identify all members of a class as having the same set of attributes (and methods).

Constraints are used to restrict the otherwise unrestricted extensions of the class diagram elements. Constraints provide an essential means of knowledge engineering, since they extend the expressivity of diagrams.That is:

- **Class and association constraints**: restrict the set and relationship extensions of classes and associations, respectively.
- **Attribute constraints** restrict attribute values in terms of types and multiplicity.
- **Binary association multiplicity constraints**: A cardinality constraint (also termed multiplicity constraint) imposed on a binary association A between classes C_1 and C_2 with roles rn_1, rn_2, respectively, is symbolically denoted:

$$A(rn_1 : C_1[\ min_1,\ max_1], rn_2 : C_2[\ min_2,\ max_2]) \tag{2.1}$$

The multiplicity constraint $[min_1, max_1]$ that is visually written on the rn_1 end of the association line is actually a participation constraint on instances of C_2. It states that an instance of C_2 can be related via A to n instances of C_1, where n lies in the interval $[min_1, max_1]$. For example, according to Figure 2.1, an *Academic* can advise at least two *Graduate*s (as indicated by the 2..* multiplicity constraint).

– **N-ary association multiplicity constraints**: Multiplicity constraints are set in n-ary association R between the classes $C_1, ..., C_n$ and the roles $Rn_1, ...Rn_n$ respectively is symbolically denoted by the following relationship construct:

$$R(Rn_1 : C_1[m_,n_1], ..., Rn_n : C_n[m_n, n_n]) \tag{2.2}$$

Multiplicity constraint on n-ary association end (role) $[min_i, max_i]$ represents the possible number of values (objects) of C_i , when the values at the other n-1 ends are fixed. Consider the ternary association in Figure 2.1, A *Student* will not take the same *Course* from more than one *FacultyMember*, but a *Student* may take more than one *Course* from a single *FacultyMember*, and a *FacultyMember* may teach more than one *Course*. The cardinality constraint defined in the binary association is clear and is set on all class instances. Conversely, as it was shown in [34], this is not the case for n-ary associations, where the cardinality constraint, as defined by UML documents, is incomplete and unclear and allows more than one possible interpretation[1]. However, The authors of [34] highlight important problems that crop up from both kinds:

1. *Class look across cardinality constraint*: Cardinality constraints that set on participating class objects: Given $n-1$ roles of n-ary association R, then every possible combination of values (objects) of those $n-1$ roles should satisfy the cardinality constraint on the last role. However, a minimum multiplicity of 1 or more assigned to one role forces all potential combinations of instances of the

[1] Consult Balaban and Shoval [1] for a discussion of the different semantics of the cardinality constraints.

remaining classes to actually exist within a n-tuple. In other words, for $n-1$ roles, we need the *Cartesian product* of $n-1$roles values to be related to instance or more from the other role. The authors of [34] these results: the "*bouncing effect of the one*".

2. *Relationship look across cardinality constraint*: Cardinality constraints that set on the relationship links: Given $n-1$ roles of n-ary association R, then every existing object from ***n-1 ends*** which is linked by a link within the n-ary association, should satisfy the cardinality constraint on the last role. However, In this kind a minimum constraint of 0 has no meaning. That is because if an instance of n-1 *exists* in the relationship, then it has to be related to an instance at the other side. So, in this interpretation, the minimum multiplicity is always at least 1. The authors of [34] called this result: the "*zero-forbidden effect*". This result is not consistent with the frequent assignment of minimum multiplicity of zero and with UML Semantics Document [22].

Indeed, both interpretations are possible. However, the most frequent kind (interpretation) of cardinality constraints for non-binary associations is *Relationship look across cardinality constraint*.

– **Association classes** restrict their objects to be identified by pairs of the connected association. In Figure 1, every *Enrollment* object is identified by a course-student pair (no two enrollments are identified by the same pair).

– **Qualifier attribute constraints**: are optional part of a binary association ends. It distinguishes the set of objects at the far end of the association based on the qualifier value. A qualifier is used within a *qualified association* to relate a *qualified object* to a *target object* using a *qualifier* value from a set of *qualifier* values. The multiplicity on the target side defines the number of target objects that can be related to one (qualified object, qualifier value) pair. The multiplicity on the qualified side of the relationship

can be 1 or *, representing the number of (qualified object, qualifier value) pairs to which the target object is related. In Figure 2.1, the binary association between *FacultyMember* and *Course* is qualified by the attribute *Semester*. This says, that *FacultyMember* (*qualified class*) can teach at most one *Course* (*target class*) in a given *Semester* (*qualifier*).

- **Aggregation constraint**: reflects whole-part relationships between a class- the assembly to its parts- the component classes. For example, the class *University* in Figure 2 is the assembly where the classes *MathFaculty* and *CompFaculty* are the component classes. The aggregation relationship is transitive and asymmetric across all aggregation links. The asymmetric property of aggregation requires that a part of an assembly cannot aggregate one of its aggregators (unlike associations in which associated classes are peers). Composition is a form of aggregation with more specific semantics that correspond to physical containment and various notions of ownership. This semantics allows more than one composition association for one component class with the constraint that a given object cannot be shared (an object may be part of only one composite).

- **Association constraints**: It is also possible to define explicit constraints on an association:

  1. **{xor} constraint**: connects two or more associations that are connected to a single class (the base class) at one end. An instance of the base class may participate in exactly one association connected by the constraint. The multiplicity of the chosen association must be observed [23].
  2. {**subset constraint**}: The subset constraint means inclusion of the link sets.

- **Association hierarchy constraints**: means inclusion of the link sets (the distinction between subset and specializing an association is not clearly described in the UML2 specification [23].

- **Class hierarchy constraints**: also called ISA constraints, specify subset relation between extension classes. In Figure 2.1 , the extensions of the *FacultyMember* and *Graduate* classes are subsets of the *Academic* extension.

- **Generalization set constraints**: Class hierarchy constraints can be grouped into a *Generalization Set* (shortly *GS*), as shown in Figure 1. For example, *GraduateCourse*, *UnderGraduateCourse* and *Course* form a Generalization Set. In that case, more constraints can be defined on the group. There are two orthogonal planes for defining such constraints: (1) *disjointness* and (2) *completeness*. Below are the four constraints that can be labeled the generalization set:

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

  The *GS* constraints can be combined to form one of the following valid combinations: {complete, disjoint}, {incomplete, disjoint}, {complete, overlapping}, {incomplete, overlapping}.

  The constraint citation { *overlapping, complete* } on the generalization set *Course* indicates that a course may be simultaneously a graduate and an undergraduate course citation { overlapping } , and every course is either graduate or undergraduate { complete }.

Def 2.1. A *legal instance* of a class diagram is an instance where the class and association extensions satisfy all constraints in the diagram. Correctness of a class diagram involves consistency and satisfiability notions, that are discussed in [4, 8, 18, 24]. We further elaborate

this terminology, and suggest additional notions, in order to facilitate a more accurate definition of correctness.

2.2.1 Discussion of the Semantics of the Overlapping and Disjoint Constraints

The overlapping/disjoint semantics in UML documents is confusing and may have different interpretations. The ambiguous issue involves the specification of exactly which subclasses are required to be overlapping/disjoint. The reasonable interpretations of each case involve either the overall collection of subclasses, or a pairwise restriction on subclasses. Correspondingly, we suggest two feasible interpretations for each kind of restriction:

1. **local overlapping**: Meaning there exists an overlapping pair of subclasses in a generalization set.
2. **global overlapping**: Meaning non-empty intersection of all subclasses in a generalization set.
3. **local disjointness**: Meaning every pair of subclasses in a generalization set is disjoint.
4. **global disjointness**: Meaning empty intersection of all subclasses in a generalization set.

For overlapping constraints, the global version is more restrictive, i.e., it implies the local version, while for disjointness constraints, the local version is more restrictive, i.e., it implies the global version. We think that both versions are reasonable and might be necessary, although the local version might be more popular for both constraints [23]. Therefore, we suggest to adopt the local version for each kind of constraint as the default, and require explicit specification if the global version is intended. This thesis adopt only the *Local* interpretation for both constraints.

Classification of Class Hierarchy Structures

ISA constraints can arise in various structures. We distinguish among three parameters that determine the class hierarchy structures and content:

1. **Graph Structure**: *ISA* constraints can form three graph structures:

   (a) **Tree Structure**: A subclass has only one super-class. For example: Figure 2.2

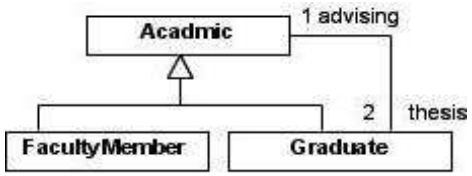


Figure 2.2: Tree Hierarchy Structure

   (b) **Acyclic Structure**: Multiple inheritance is allowed, but the undirected graph formed by the *ISA* constraints is acyclic. For example, in Figure 2.3, the hierarchy structure is not a tree, as *Graduate* is a sub-class of both *Acadmic* and *Student*, but the undirected class hierarchy graph is acyclic. The acyclic structure prevents multiple inheritance from a common ancestor-class and ensures that different ISA hierarchies with multiple inheritances overlap in at most one class.

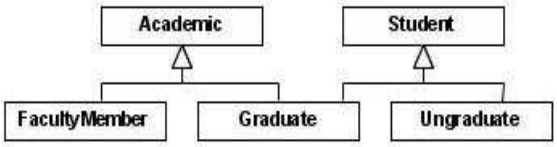


Figure 2.3: Acyclic Hierarchy Structure

   (c) **Graph Structure**: Unrestricted multiple inheritances, some of the undirected graphs formed by the *ISA* constraints is cyclic. The cyclicity in *ISA* constraints is caused by multiple inheritances from a common ancestor class (for example, Figure 2.4-a), or different ISA hierarchies that overlap in two classes or more (for example Figure 2.4-b).

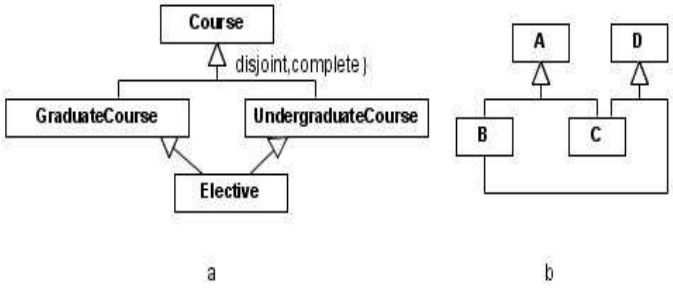


Figure 2.4: Constrained Graph Hierarchy Structure

2. **Presence of *GS* Constraints**.

3. **Number of *GS*s per superclass**: The case of multiple *GS*s per super-class is distinguished from the simpler case of a single *GS* per super-class. For example in Figure 2.1, the class Student has two generalization sets.

We use an abbreviated notation that specifies the value of these parameters. The *hierarchy structure* is denoted by one of {T,A,G}, standing for *Tree structure, Acyclic graph*, and *Graph*, respectively. The presence of *GS* constraints is denoted by *C*, and the presence of multiple *GS*s per superclass is denoted by *M*. The multiple *GS*s per superclass distinction is relevant only for tree structure hierarchies. The resulting hierarchy variants are: [T]-GS for tree structured unconstrained hierarchy with a single *GS* per superclass; [T-C]-GS for tree structured constrained *GS*s with a single *GS* per superclass; [T-C-M]-GS for a constrained tree hierarchy with multiple *GS*s per super class; [M]-GS for tree structured unconstrained hierarchy with multiple *GS*s per super class; [A]-GS for an unconstrained acyclic hierarchy; [A-C]-GS for a constrained acyclic hierarchy, ; [G]-GS for unconstrained graph hierarchy; [G-C]-GS for a constrained graph hierarchy.

2.3 Reasoning Needs in Class Diagrams

Class diagrams are models written by people, and therefore, usually suffer from modeling problems like inconsistency, redundancy and abstraction errors. Inexperienced designers

tend to create erroneous models, but even experienced ones cannot anticipate the implication of a change on an overall model [29]. Indeed, Lange et al showed in [20] that model defects often remain undetected, even if experienced practitioners check the model attentively. These problems are aggravated when a model originates from different resources, as frequently happens when Web services are integrated. Combined sources are usually overlapping, and the integration yields redundant inconsistent models [35, 12, 3, 28]. It is a common belief that such problems can best be solved at the level of models [27].

Thus, the need to provide coherent models is appealing. In particular, in order to avoid inconsistencies, unsatisfiability, redundancies, and ambiguities, it is essential to have tools that can support validation of the models. Furthermore, models can be improved, based on given design criteria. For ontology engineering purposes, it is of the utmost importance to keep the model accurate, consistent, and unambiguous, as it usually serves multiple knowledge bases. The same holds for meta models as they underlay the modeling of concrete systems. In order to achieve the goal of improving a model quality, a diversity of reasoning capabilities is required.

Reasoning with formalism requires the existence of formal semantics that assigns an exact meaning to each expression of the formalism. Indeed, there is a wide research effort whose goal is to provide formal semantics to class diagrams (and to UML as a whole). One way to define meaning is to directly assign a denotation to every language construct. This is termed declarative direct semantics. An alternative way to define language semantics is to use a different formally defined language that already has a formal semantics, as a "mediator". That is, instead of defining the meaning of elements declaratively as described above, they are translated into expressions of the intermediate language. This approach is termed indirect semantics.

The semantics of UML class diagrams has been defined using both approaches. The direct semantics of class diagrams assigns set extensions to classes and associations. It has been formally defined by several studies such as [36, 31]. There are several indirect semantics

approaches for class diagrams in the literature. Full first order logic is the most popular intermediate language for class diagrams without OCL constraints [4, 32] or with OCL constraints [37, 21]. Some other intermediate languages are Z and Object-Z [38, 39, 40, 40, 33] , Algebraic specifications in the form of Abstract Data Types [41, 42, 10], Hierarchical Predicate Transition Nets [43] and B specification [46, 44].

In view of the wide spread usage of UML class diagrams and the difficulties of producing high quality models, it is essential to equip UML case tools with reasoning capabilities. The additional power can help with detecting design problems, identifying the reasons for these errors, suggesting possible solutions and providing advice for design improvements [30, 45, 4, 19].

2.3.1 Class Diagram Reasoning Problems

Reasoning on UML models in general and on class diagrams, in particular, gains much attention, recently. Reasoning is necessary for improving the design quality and for supporting application construction needs. Design quality refers to (1) erroneous models that impose inconsistent constraints, (2) redundant models that can be simplified, and (3) models that can be improved according to some design criteria [14, 4]. Reasoning helps in detecting erroneous models, finding the source of errors and possibly suggesting repairs. It is used for revealing redundant situations, and for testing whether design criteria are met.

Questions about class diagram quality deal with inconsistency, finite satisfiability, redundancy, and design improvement. Inconsistency arises when the constraints imposed by a class diagram are contradictory. Finite satisfiability is caused by multiplicity constraints that can be satisfied by either empty or infinite class extensions (i.e., instantiations). Redundancy appears when constraints seem to allow values or links that cannot be realized (are inconsistent). Quality improvement deals with changing the models following various criteria such as design patterns or reuse enhancements.

Inconsistency and lack of finite satisfiability, are considered erroneous design. The first,

because an inconsistent class diagram does not have a non-empty extension, and the latter, because there is no finite and non-empty extension [9]. For example, constraints that imply simultaneous disjointness and non-empty intersection of classes cause inconsistency (contradiction). Both of these problems decrease the quality of a UML model, since they reflect a symptom of bug in the analysis phase [9]. They might delay system development and increase its cost [19]. [20] shows that defects often remain undetected, even if the model is read attentively by practitioners. Most reasoning efforts have been devoted to the identification, detection and repair of inconsistency.

In the next sections we follow [18, 25, 8] and present the finite satisfiability notion. Although this work focuses only on the satisfiability problems, for the sake of completeness we present also the consistency notion. These notions are based in [4]. However, we suggest additional notions for both satisfiability and consistency notions. These additional notions shall contribute to more accurate and comprehensive investigation of reasoning problems. For other analogous terminologies we refer the reader to [11, 16] .

2.3.2 Inconsistency Problems

Inconsistency means class emptiness, that is, a class cannot be instantiated since the constraints imposed on its instances cannot be satisfied. The extreme case involves all classes in the diagram. . Indeed Berardi et al distinguish in [4] between two cases:

- ***Consistency of a class diagram***: A class diagram is *consistent, (satisfiable)* if it has an instantiation with at-least one non-empty class extension. Otherwise, it is inconsistent .

- ***Class consistency***: A class is *consistent* if there is an instantiation in which the class extension is non-empty. Otherwise, it is *inconsistent, (unsatisfiable).*

We claim that class diagram consistency is of less importance, since every class diagram that has an unconstrained class (through an association) is consistent. Figure 2.5 presents

an example of such a diagram. The Seminar class is totally unconstrained and can be freely instantiated.

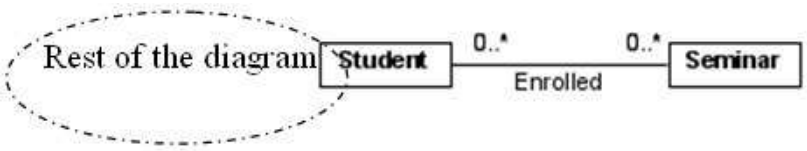


Figure 2.5: Unconstrained class diagram

In fact, every realistic class diagram that we have checked proved to have such a class. Consequently, we suggest two additional notions termed *"all class consistency"* and *"full consistency"*:

- ***All class consistency of a class diagram***- A class diagram is all class consistent if every class is *consistent*.
- ***Full consistency of a class diagram***- A class diagram is fully consistent if it has an instance in which all class extensions are non-empty.

full consistency implies *all class consistency*, which implies consistency. *Consistency of a class diagram* does not imply *all class consistency*. It is not clear whether *all class consistency* implies *full consistency*, that is, whether the existence of a non-empty instantiation for every class implies the existence of a single instance of an object diagram in which all class extensions are non-empty.

Inconsistency means class emptiness, that is, a class cannot be instantiated since the constraints imposed on its instances cannot be satisfied. Inconsistency is caused by constraint contradiction. UML class diagrams include seven kinds of constraints: Class hierarchy relations, generalization set constraints, multiplicity constraints, attribute multiplicity, association class constraints, association constraints and aggregation. Kaneiwa and Satoh identify in [32] some factors for class inconsistency. We extend their work and characterize contradictory interactions between constraints.

1. ***Inconsistency due to generalization set constraints***:

   Such inconsistency occurs when, for example, a generalization hierarchy has a disjoint constraint, where the disjoint subclasses have a common subclass. Figure 2.6(a) presents a case in which the disjointness constraint forces the *Elective* class to be empty. In Figure 2.6(b), the interaction of the disjointness and the completeness constraints forces class B to be empty. Any instance of E must be an instance of one of subclasses C or D, but classes B, C and D satisfy the *disjointness* constraint.

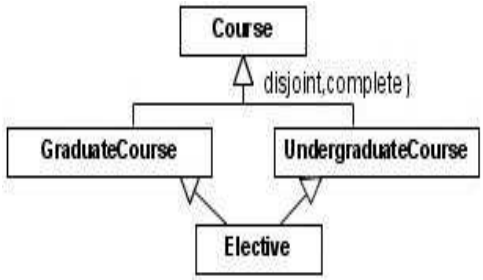


Figure 2.6: Inconsistency due to generalization-set constraints

2. ***Inconsistency due to the interaction between class hierarchy constraints and multiplicity constraints***: Such inconsistency may occur when contradictory multiplicity constraints exist among associations that are related by a subset constraint. In Figure 2.7 the multiplicity constraint of class *Member* within the *LargeComposition* is 5. Yet, as this association is defined as a subset of the *Composition* association which has multiplicity constraint 1..3, there is a contradiction between these constraints, implying a necessarily empty extension for class *Large*.

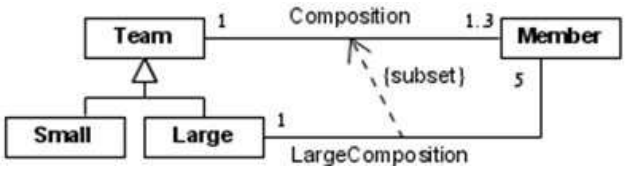


Figure 2.7: Inconsistency due to class hierarchy and multiplicity constraints

3. ***Inconsistency due to the interaction between association constraints and***

multiplicity constraints: XOR constraint between associations contradict a non-zero minimum multiplicity constraint since the XOR implies that only one of the associations is realized in every instantiation. Figure 2.8 represents such a situation.

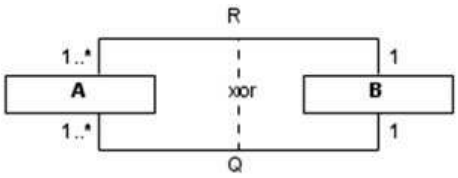


Figure 2.8: Inconsistency due to multiplicity and inter-association constraints

2.3.3 Finite Satisfiability in Class Diagrams

Finite satisfiability means that a class has a finite non-empty extension. Lack of finite satisfiability is certainly erroneous since it means that the class cannot be instantiated. An example of a finite satisfiability problem can be derived from Figure 1 in Section 2. In this example, the cardinally and the class hierarchy constraints between classes *Graduate* and *Academic* enforce the *Graduate* class to be either empty or infinite.

Finite satisfiability is an essential property for all classes, since a necessarily empty or infinite class extension means that a class is either empty, and hence redundant, or its instantiation causes infinite loops. Similarly to the inconsistency case, finite satisfiability can also refer to a single class or to the whole class diagram. However, while class diagram consistency requires the consistency of at least one class, finite satisfiability must apply to all classes.

In analogy with the terminology for inconsistency, we introduce three terms: *Class finiteness*, *all class finiteness*, and *full finiteness*.

- ***Class finite satisfiability (Class Finiteness)***: A class is finitely satisfiable if there is a finite instance of the class diagram, in which the class extension is non-empty. A class diagram instance is finite if all class extensions are finite.

- **All class finite satisfiability of a class diagram (all class finiteness)** - A class diagram is all class finitely satisfiable if for every class there is a finite instance in which the class extension is non-empty. Lenzerini and Nobili [18] used the notion strong satisfiability for this term.
- **Full finite satisfiability of a class diagram (full finiteness)** Full finite satisfiability of a class diagram - A class diagram is fully finitely satisfiable if it has a finite instance in which all class extensions are non-empty.

Clearly, full finite satisfiability implies all class finite satisfiability. The inverse is also true, i.e., all class finite satisfiability implies full finite satisfiability. Lanzerini and Nobili [18] show that for class diagrams with multiplicity constraints alone. In Chapter 3, we extend this result to apply to unrestricted class diagram. The theorem is proved by the following argument: Every two disjoint instances can be combined into a single instance of the class diagram. The argument holds due to the special character of UML class diagram constraints, which are closed under disjoint instance combination.

Finite satisfiability problems are caused by cycles of conflicting multiplicity constraints. The cycles might include class hierarchy constraints, generalization set constraints, and association class constraints. Eight sets of conflicting constraints can prevent finite satisfiability and thus give rise to infiniteness problem. This thesis focuses on these problems arising from the following four cases: *multiplicity constraint conflict*, *ISA and multiplicity constraint conflict* and *multiplicity constraint and generalization set constraints conflict*. The rest fall outside the scope of this research. However, for the sake of providing a complete picture we will describe below all eight cases that give rise to finite satisfiability problems. Examples are provided for cases 1-6. The remaining three will be discussed in depth later in this thesis.

1. ***Lack of finite satisfiability due to cardinality constraint conflict:***

   In Figure 2.9, each course should have a single successor and at least two predecessors. Therefore, if the number of courses in a diagram instance is C, and the number of

Dependency links is D, then D must satisfy $D = C \cdot 1$ and $D \geqslant C \cdot 2$, implying the inequality: $C \geqslant C \cdot 2$, that can be satisfied only by empty or infinite extensions for class *Course*. Therefore, although class *Course* is consistent it is not finitely satisfiable.

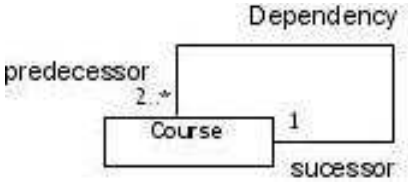


Figure 2.9: Unsatisfiability due to multiplicity constraint conflict

2. ***Lack of finite satisfiability due to ISA and multiplicity constraint conflict*:**

   Figure 2.10 presents a multiplicity constraint cycle that involves a compound class, *Gradaute*, whose instances must be related to *Academic* instances. Therefor, using similar considerations as in the previous case, the number of *student-advisor* links in every diagram instance must be both, $G \cdot 1$ and $A \cdot 2$, assuming that G and A are the number of graduates and academics, respectively. Therefore, the extensions of *Graduate* and *Academic* must satisfy $G = A \cdot 2$, while the *Graduate* extension is a subset of the *Academic* extension. This constraint can, again, be satisfied only by empty or infinite extensions.

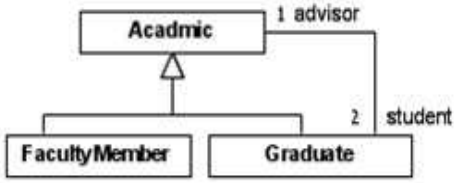


Figure 2.10: Unsatisfiability due to class hierarchy constraint conflict

3. **Lack of finite satisfiability due to interaction between an association class constraint and multiplicity constraints**:

   Figure 2.11 presents a case where an association class, *Contract*, is constrained by contradictory multiplicity constraints. In every legal instance, the number C of con-

tracts is as twice the number of employees, E, since every employee is linked with two departments. Yet, the number of contracts is equal to the number of employees, as dictated by the *Manager* association. That is, $C = E \cdot 2$, and $E = C$, implying $E = E \cdot 2$ which can be achieved only by either empty or infinite extensions for all three classes.

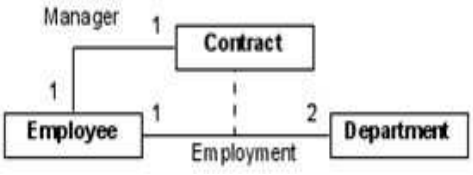


Figure 2.11: Unstatisfiability due to association class and multiplicity constraints.

4. ***Lack of finite satisfiability due to multiplicity constraints conflict that is caused by class and association hierarchy***

   In Figure 2.12, classes Member and Large are related via associations *Management* and the multiplicity constraints in these associations can be finitely satisfied. However, the *LargeComposition* association is constrained to be a subset of the *Composition* association between Team and Member, implying that the multiplicity constraint of Member in *LargeComposition* is actually tightened into 2..3. This restricted multiplicity range causes a finite satisfiability problem. For the *Management* association, if M, L and Man are the number of instances of Member, Large and *Management*, respectively, then $Man = L \cdot 4, Man = M \cdot 1$ imply $M = L \cdot 4$. For the *LargeComposition* association, if LC is the number of its links, then $L \cdot 2 \leqslant LC \leqslant L \cdot 3$, $LC = M \cdot 1$ imply $L \cdot 2 \leqslant M \leqslant L \cdot 3$. Replacing M by $L \cdot 4$ yields the inequality $L * 2 \leqslant L \cdot 4 \leqslant L \cdot 3$, that can be satisfied only if the Large, and thereby the *Member*, extensions are either empty or infinite.

5. ***Lack of finite satisfiability due to asymmetry of aggregation:***

   The asymmetric property of aggregation requires that a part of an assembly cannot

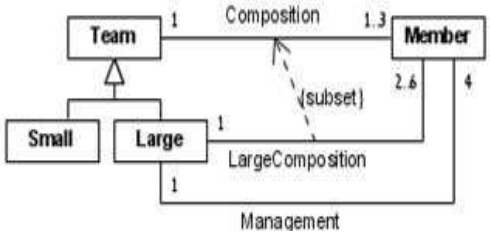


Figure 2.12: Unstatisfiability due to subset and multiplicity constraints.

aggregate one of its aggregators. Consequently, a legal instance of a class diagram cannot include aggregation cycles. Figure 11 depicts a case, where an organization consists of at least one child organization. Due to the acyclic character of the aggregation association, this multiplicity constraint can be satisfied by either an empty or an infinite chain of organizations.

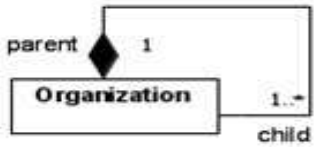


Figure 2.13: Unsatisfiability due to the asymmetry of aggregation.

6. ***Lack of finite satisfiability due to multiplicity constraints conflict that is caused by look-across multiplicity constraints on n-ary association and multiplicity constraints on binary association:*** N-ary association with *class look across cardinality constraint* is not used as widely as n-ary association with relationship *look-across cardinality constraint*. However, as we pointed out earlier in this chapter, n-ary associations with *class look across cardinality constraint* is valid and consistent with UML semantic despite being less common.

   The class diagram in Figure 2.14 presents a ternary association with look-across cardinality constraints. Consequently, every combination of objects of A and B should be related to one instance of C, every combination of objects of A and C should be related to one instance of B and every combination of objects of B and C should be

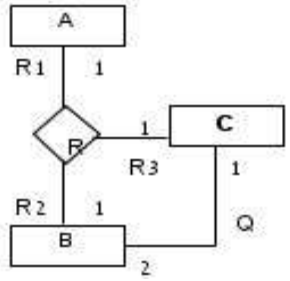


Figure 2.14: Class Look-Across: Infiniteness Problem

related to one instance of A. Yet, if the number of instances of A, B, C is a, b and c respectively and if the number of R links (tuples) is r. Then r must satisfy: $r = a \cdot b$, $r = a \cdot c$ and $r = c \cdot b$. Therefore: $a = b = c$. But the cardinality constraint within the binary association Q dictates that: $b = 2c$. Hence, the above class diagram can be satisfied only by empty or infinite extensions.

7. ***Lack of finite satisfiability due to generalization set constraint:*** Will be discussed in depth later in this thesis.

2.3.4 Methods for Reasoning

Class diagram reasoning methods can be classified into *concrete reasoning methods* that directly solve specific problems [18, 25, 11, 32] and *translation based methods* that provide reasoning by mapping UML models into a formal reasoning framework [4, 42, 44] [2]. Concrete methods tend to apply to error detection and revealing redundancy, while translation based methods deal with general query answering a variety of modeling needs.

The main advantage of the translation-based approach is the uniform handling it provides for a variety of problems. Once the class diagram is translated, such methods rely on an already existing reasoner for question answering. Therefore, a single translation can serve for

[2] A UML class diagram is translated into a formula or expression in some other language, and the translation is proved to be correct. The notion of correctness varies between studies. The formal notion requires a proof of equivalence, i.e., a proof that the translation preserves all and only the implications of the original class diagram.

answering many questions. On the other hand, they cannot optimize solutions to problems.

Concrete methods present an opposite approach. Each method is usually targeted at solving a single problem, in an optimal way. Consequently, solutions are better dedicated to solve their problems, but many such solutions need to be designed. Yet, the concrete methods tend to scale up to large problems better, and it is easier to embed such algorithms within UML case tools. This option seems unreasonable for translation-based methods, as the embedding of a full reasoner tends to be quite heavy.

When reasoning about various aspects of consistency, researchers have distinguished three levels of a solution: problem detection, cause identification, and repair [16]. Problem detection means just notification that a problem exists. Cause identification means detecting the reason for the problem, and repairing amounts to suggesting a solution.

In this work, we adopt the *concrete reasoning methods* approach and suggest several finite reasoning methods for deducting finite satisfiability problems in restricted UML class diagrams. In the following sections, we investigate at length previous works which adopt the concrete approach for finite reasoning. But first, for the sake of completeness, we present briefly previous works which had adopted the *translation based methods* approach and the work of [32, 48] which adopt *concrete reasoning methods* for reasoning about emptiness of class diagrams.

Translation-Based Methods for Reasoning about Class Diagrams

Berardi et al. [4] encode class diagrams in description logics and prove that reasoning about UML class diagrams is EXPTIME-hard. They also show that under minor restrictions, UML class diagrams can be encoded in the description logic $ALCQI$ which is the most expressive description logic that is supported by DL reasoning tools. [50] describes a tool that translates UML class diagrams into the DL reasoners FaCT and Racer. The ICOM tool of [49] supports DL based reasoning on Extended Entity Relationship and UML class diagrams. The encoding of UML class diagrams into Racer suggests that UML case tools

can provide reasoning capabilities by association with DL reasoners. In fact, apart from finite satisfiability, all reasoning problems discussed in this paper can be answered using the DL translation. Moreover, using the DL encoding, all reasoning problems can be reduced to the emptiness problem.

A different translation based approach for reasoning about class diagrams is reported in Andre et al. [41, 42]. In this experiment, UML class diagrams are translated into abstract data types, and the Larch prover [47] is used for reasoning about them. Similarly to the description logics based translation, the method enables reasoning on a variety of inconsistency problems. The problem seems to lie in scaling, as each class diagram element translates into multiple algebraic rules, and the prover cannot handle large sets of rules.

Concrete Methods for Reasoning about Emptiness of Class Diagrams

Kaneiwa and Satoh [32] study the problem of full consistency in a subset of UML class diagrams that include classes with typed attributes and multiplicity constraints on the attributes, unconstrained associations and constrained generalization sets. They identify three factors for inconsistency in such diagrams: (1) combination of generalization with disjointness; (2) attribute overwriting in multiple hierarchies; and (3) combination of completeness and disjointness constraints in generalization sets. Based on these factors, they provide tractable algorithms for deciding full consistency in the restricted class diagram model. This method is implemented as a debugging system for restricted class diagrams [48].

Concrete Methods for Reasoning about Finiteness of Class Diagrams

Reasoning on finiteness of entity relationship and class diagrams has attracted much attention. The problem was independently identified in (Lenzerini and Nobili: [18]) and in (Thalheim: [26, 25, 24]), and referred to entity relationship diagrams. Later the methods were extended to various fragments of UML class diagrams. The problem is to detect, identify cause and suggest repair, to diagrams that are not strongly satisfiable.

There are two main approaches: (1) The linear programming approach, and the (2) graph based approach. The first approach reduces the *all class finiteness* problem to the problem of finding a solution to a system of linear inequalities. The second approach detects infinity causing cycles in the diagram, and possibly suggests repair transformations. All methods apply only to fragments of UML class diagrams. Detection of infinity in unrestricted UML class diagrams is still an open issue.

The Linear Programming Approach The fundamental method of Lenzerini and Nobily [18] is defined for an entity relationship diagram that includes Entity types (Classes), n-ary Relationship types (Associations), and Cardinality Constraints[3]. The method consists of a transformation of the cardinality constraints into a into a set of linear inequalities whose variables stand for the sizes (cardinalities) of the entity and relationship types in a possible instance. A relationship R in Figure 2.15 yields the four inequalities 2.3.

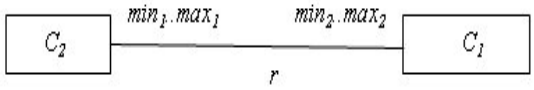


Figure 2.15: Binary Association

$$r \geqslant min_2 \cdot c_1, \;\; r \leqslant max_2 \cdot c_1, \;\; r \geqslant min_1 \cdot c_2, \;\; r \leqslant max_1 \cdot c_2 \tag{2.3}$$

where r, C_1, C_2, are variables that stand for the sizes of the respective entity or relationship types. In addition, For every entity or association symbol T, insert the inequality:

$$T > 0 \tag{2.4}$$

The size of the inequality system is polynomial in the size of the diagram. The main

[3]Lenzerini and Nobili (1990) use the membership semantics for cardinality constraints (consult Balaban and Shoval in[1, 2] for semantics of cardinality constraints) for semantics of cardinality constraints). For non-binary relationships, this is not the standard semantics of cardinality constraints, neither in the entity relation model nor in the class diagram model.

result is that the entity relationship diagram is fully finitely satisfiable if and only if the inequalities system has a solution. Since linear programming is solvable in polynomial time in the size of the problem encoding, full finite satisfiability for this fragment of class diagrams can be decided in polynomial time.

Example 2.1. *Consider Figure 2.9, each course should have a single successor and at least two predecessors. The applying of Lenzerini and Nobily method in this example yields the insolvable inequalities system below:*

1. ***The Variables**: c for Course and d for Dependency*
2. ***The System Inequalities**:*
   (a) ***The Dependency Association Inequalities (2.3)**:*
      - *1.* $d \geqslant c * 2$.
      - *2,3.* $d = c$, $(d \geqslant c \quad and \quad d \leqslant c)$.
   (b) *4,5.* $d, \quad c \; > \; 0$

Calvanese and Lenzerini, in [8], extend the inequalities based method of [18] to apply to schemata with class hierarchy constraints. The expansion is based on the assumption that class extensions may overlap. They provide a two stage algorithm in which the finite satisfiability problem of a class diagram with *ISA* constraints is reduced into the finite satisfiability problem of a class diagram without *ISA* constraints. Then, similarly to [18], they check satisfiability of the new class diagram by deriving a special system of linear inequalities (different from that of [18]).

The class diagram transformation process of [8] is fairly complex, and might introduce, in the worst case, an exponential number, in terms of the input diagram size, of new classes and associations. The method was further simplified in [9], were class overlapping is restricted to class hierarchy alone. The simplification of [9] reduces the overall number of new classes

and associations, but the worst case is still exponential. The following example presents the application of [9] to Figure 2.10.

Example 2.2. *The application of the [9] method yields four classes and eight associations. Each class and association is represented by a variable in the resulting inequalities system. The variables are:*

1. ***Class variables:*** a_1 *for an Academic that is neither a Graduate nor a FacultyMember;* a_2 *for an Academic that is a Graduate but not a FacultyMember;* a_3 *for an Academic that is a FacultyMember but not a Graduate;* a_4 *for an Academic that is simultaneously a Graduate and a FacultyMember;*

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

The inequalities system below results from application of the method of [9] to Figure 2.10. Equations 1-4 translate the 2..2 multiplicity, equations 5-6 translate the 1..1 multiplicity, and the inequalities in 7-9 represent the satisfiability conditions. The inequalities system unsolvable, implying that the class diagram in Figure 2.10 is unsatisfiable.

1. $2a_1 = ad_{12} + ad_{14}$
2. $2a_2 = ad_{22} + ad_{24}$
3. $2a_3 = ad_{32} + ad_{34}$
4. $2a_4 = ad_{42} + ad_{44}$
5. $a_2 = ad_{12} + ad_{22} + ad_{32} + ad_{42}$

6. $a_4 = ad_{14} + ad_{24} + ad_{34} + ad_{44}$

7. $a_1, a_2, a_3, a_4, ad_{12}, ad_{14}, ad_{22}, ad_{24}, ad_{32}, ad_{34}, ad_{42}, ad_{44} \geqslant 0$

8. $a_1 + a_2 + a_3 + a_4 > 0$,

9. $ad_{12} + ad_{14} + ad_{22} + ad_{24} + ad_{32} + ad_{34} + ad_{42} + ad_{44} > 0$

Boufares and Bennaceur [3] suggest using the Fourier-Motzkin elimination method [56] for solving the obtained system of linear inequalities. They show how this method can help in identifying the source of infinity, when the inequalities system is unsolvable. The idea is that backtracking the solution process reveals the conflicting cardinality constraints.

Lenzerini and Nobili [18] were the first to suggest a method for cause identification of strong satisfiability in restricted entity relationship diagrams. Their solution is not constructive, as they do not provide a method for computing critical cycles. A first step towards finding critical cycles appears in [25].

Dullea and Song [51] and Dullea et al.[52] characterize infinity causing structures (termed structural invalidity) of recursive binary and ternary relationship types in entity relationship diagrams. The analysis suggests a set of structure based decision rules for identifying structural invalidity in entity relationship diagrams.

Graph Based Approach Hartman [11, 16] handle the problems of full strong satisfiability from all three aspects of detection, cause identification and repair. Hartman [11] suggests a polynomial time graph-theoretical method for the detection of full strong satisfiability in entity relationship diagrams (same model as in Lenzerini and Nobili [18]). He defines a similar notion of critical cycles, and uses it for detecting infinity problems. In addition, for graphs without critical cycles, the method can derive a minimal finite instance. More details on this work appear in [12, 13] and in [15].

Hartman suggests in [16] critical cycles based methods for cause identification and repairing strong satisfiability problems. The paper suggests four heuristic strategies for repairing

infinity problems. One method is based on finding minimal inconsistent constraint sets that exist in every critical cycle, and suggests how to repair them. Another method is based on the notion of feedback arc set which is a set of arcs that intersects all critical cycles in the graph. The paper suggests a cardinality constraint repair plan, based on an optimization method for finding a minimal feedback arc set. Hartmann extends in [14, 17] the former work for reasoning about a set of cardinality constraints, key constraints, soft constraints and functional dependencies. He presents a list of seven implication rules for deriving new cardinality constraints from given ones.

Scope of The Thesis

This thesis focuses on class Diagram with: binary association, unrestricted class hierarchy structure without generalization set constraints (all the {[T], [M], [A], [G]}- Structures) and acyclic hierarchy structure with possible generalization set constraints (all the {[T-C], [M-C], [A-C]}-Structures). This thesis adopts the restricted interpretation for *overlapping* and *incomplete*, which implies the existing of at least one common instance for *overlapping* and at least one instance beyond the subclasses extensions for the *incomplete* constraint.

The next chapters presents our finite reasoning methods on UML class diagram within the scope of the thesis. Our methods adopt the linear programming approach and constitute an extensions of the work of [18] to other components of UML class diagram.

Chapter 3

Unconstrained Generalization Sets Extension

In chapter 2, we discussed the importance of constraints in class diagrams. It was shown that constraints provide an essential means of knowledge engineering since they extend the expressiveness of diagrams. *ISA* constraints significantly enrich the modeling capabilities. They have always received a great attention in database modeling and knowledge representation. Their importance have increased with the emergence of the object oriented modeling language UML where classes are organized in a hierarchy based on the generalization principle [8]. However, due to the expressiveness of the *ISA* constraints it turns out that a model may turn vacuous because it might impose constraints that cannot be satisfied. This is due to the fact that constraints from different classes interact with each other. As a result, conflicts might raise among these constraints, leading to finite satisfiability problems.

Finite satisfiability problem in the presence of class hierarchy constraints is caused by cycles of conflicting multiplicity constraints that include also ISA constraints or generalization set constraints. Figure 3.1 shows a situation where infinity results from class hierarchy and multiplicity constraint conflict. The number of *student-advisor* links in every diagram instance must be both, $G*1$ and $A*2$, assuming that G and A are the number of graduates

and academics, respectively. Therefore, the extensions of Graduate and Academic must satisfy $G = A * 2$, while the Graduate extension is a subset of the Academic extension. This constraint can, again, be satisfied only by empty or infinite extensions.

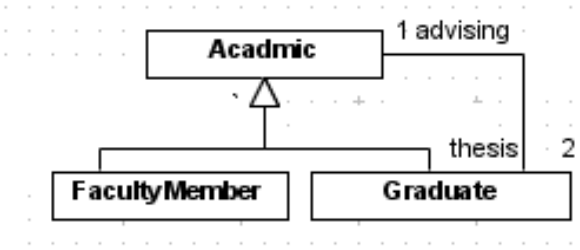


Figure 3.1: A Class Diagram with a Finite Satisfiability Problem

In this chapter, we present a method for finite reasoning of UML class diagrams that include binary associations and unconstrained class hierarchy structures. The method builds on the Lenzerini and Nobili [18]-algorithm (see Section 2.3.4), which translates a restricted ER diagram to an inequality system and tests for the existence of a solution. We start with a tree structured hierarchy [T]-GS and extend it to the rest of the hierarchical structures: {[T-M],[A],[G]}-GS.

3.1 Finite Reasoning Method over Class Diagrams with Unconstrained Tree Hierarchy Structure

In this section, we develop a method for testing the *all class finiteness* of class diagrams with unconstrained tree structured hierarchy [T]-GS. First, we state that all class finite satisfiability implies a full finite satisfiability.

Theorem 3.1. *If an unrestricted class diagram CD with unconstrained generalization sets and n-ary association with the standard interpretation of cardinality constraints (look-across cardinality constraints) is all class finitely satisfiable, then it is fully finitely satisfiable.*

Proof. Assume that CD is all class finitely satisfiable. Then, for every class in CD, there is

a non-empty finite instance in which the class extension is non-empty. Let $C_1,,,C_n$ be all classes in CD and let $I_1,,,I_n$ be non-empty and finite disjoint instances of CD in which C_i has a non-empty extension in I_i. Define, $I = \bigcup_{i=1}^{n} I_i$

Auxiliary Claim. I is a legal, finite instance of CD. That is:

1. I is finite and has non-empty extensions for all classes.
2. I satisfies all constraints in CD.

First, we state the following lemma:

Lemma 3.2. *Let CD be an unrestricted class diagram with unconstrained generalization sets and n-ary association with the standard interpretation of cardinality constraints. Let I_1 and I_2 be two disjoint non-empty instances of CD, with the possibility of the existence of one class or more of CD whose instance is empty in I_1 or I_2. Let, $I = I_1 \bigcup I_2$, then:*

1. *I is finite and non-empty and every class in CD, which has a non-empty extension in I_1 or I_2, also has a non-empty extension in CD.*
2. *I satisfies all the constraints of CD.*

Proof of Lemma 3.2.

Proof of 1: I_1 and I_2 are finite and non-empty instances. Therefore, their union $I_1 \bigcup I_2$ is so. Yet, assume, without losing generality that C has a non-empty extension in I_1. $C^I = C^{I_1} \bigcup C^{I_2}$ and $C^{I_1} \neq \emptyset$, implying $C^I \neq \emptyset$.

Proof of 2: We show that for each constraint in CD. That is:

1. **Cardinality Constraints on an Unqualified Binary Association**: Consider an association R in CD, with the classes C_1 and C_2. Assume, $R^I \neq \emptyset$. The cardinality constraints of R are satisfied separately by the R-links in I_1 or I_2 (at least one). From the disjointness of the instances I_1 and I_2, it follows that for an object $a \in C_1^I \Rightarrow a \in C_1^{I_1} \wedge a \notin C_1^{I_2}$ *or* $a \in C_1^{I_2} \wedge a \notin C_1^{I_1}$. Let us assume without losing generality

that $a \in C_1^{I_1} \wedge a \notin C_1^{I_2}$. By definition, the instance I_1 satisfies all the constraints. Therefore, for the object a, its links satisfy the cardinality constraint on R. Hence, all R-links in I satisfy the cardinality constraints.

2. **Association class constraints**: Association classes restrict their objects to be identified by pairs of the connected associations. Therefore, the union of disjoint instances (links) trivially preserves these constraints.

3. **Aggregation constraints**: Aggregation carries the semantics as binary association. Hence, the same argument that applies to the binary association holds true here. The additional constraints, which are imposed by the aggregation (asymmetric constraint and composition constraint), are also preserved in I due to the disjointness of the union of instances I_1 and I_2.

4. **Standard Cardinality Constraints on n-ary Associations**: Consider n-ary association R

   in CD, where $R^I \neq \emptyset$. For an arbitrary n-1 ends on R, let us choose n-1 values (objects), $c_1, ..c_{i-1}, c_{i+1}, ..c_n \in R^I|_{C_1,..C_{j-1},C_{j+1},...,C_n}$ where $c_j \in C_j^I$. By definition, these n-1 values (objects) $c_1, ..c_{i-1}, c_{i+1}, ..c_n$ belong to either I_1 or I_2. Therefore, the n-1-tuple $(c_1, ..c_{i-1}, c_{i+1}, ..c_n)$ satisfies the cardinality constraint of the other end (C_i). Hence, I satisfies the cardinality constraint of R.

5. **Cardinality Constraints on Qualified Associations**: Let R be a qualified association in CD with a non-empty extension in I. Therefore, a pair of a (qualified object, qualifier value) in R^I, belongs to either I_1 or I_2. Hence, its qualified links satisfy the qualified cardinality constraint on R. The same argument holds true also for *target-class* objects of R in I.

6. **ISA Constraints**: Let C be a super-class in CD and let C_1 be its subclass with a non-empty extension in I. Therefore, C_1 has a non-empty extension on at least one

instance I_1 or I_2. ${C_1}^I = {C_1}^{I_1} \cup {C_1}^{I_2}$ and both I_1 and I_2 satisfy ISA constraints. Therefore, ${C_1}^{I_1} \cup {C_1}^{I_2} \subseteq C^{I_1} \cup C^{I_2}$. But $C^I = C^{I_1} \cup C^{I_2}$, implying, ${C_1}^I \subseteq C^I$.

End of the Proof of Lemma 3.2.

Proof of Auxiliary Claim

Proof by induction on the number of union instances.

Basis:

- $k = 1$, $(I = I_1)$: according to the assumption I_1 is a non-empty legal instance of CD in which in which C_1 has a non-empty extension in I_q.
- $k = 2$, $I = I_1 \cup I_2$: Follows from Lemma 3.2.

Inductive step: $k = n$, $I = \bigcup_{i=1}^{n} I_i$: Let, $I' = \bigcup_{i=1}^{n-1} I_i$. Therefore, $I = I' \cup I_n$. By the induction's assumption, I' is finite and has non-empty extensions for all classes $C_1, ..C_{n-1}$ in CD and satisfies all constraints. Hence, by Lemma 3.2, $I = I' \cup I_n$ is finite and non-empty for all classes, which satisfies all constraints in CD. □

Algorithm 3.1. *Finite Reasoning Method for Class Diagram with [T]-GS*

- ***Input**: A class diagram CD that includes binary associations and unconstrained tree generalization sets, with at most a single generalization set for a superclass.*
- ***Output**: True, if the CD is fully finitely satisfiable; false otherwise.*
- ***Method**:*
  1. *Class diagram reduction - Create a new class diagram CD' as follows:*
     (a) *Initialize CD' by the input class diagram CD.*
     (b) *Remove from CD' all generalization set constructs.*
     (c) *For every removed generalization set construct, create new binary association between the superclass and the subclasses, with 1..1 participation constraint for the subclass (written on the super-class edge in the diagram) and 0..1 participation constraint for the super-class.*
  2. *Apply the Lenzerini and Nobili algorithm to CD'.*
- ***END***

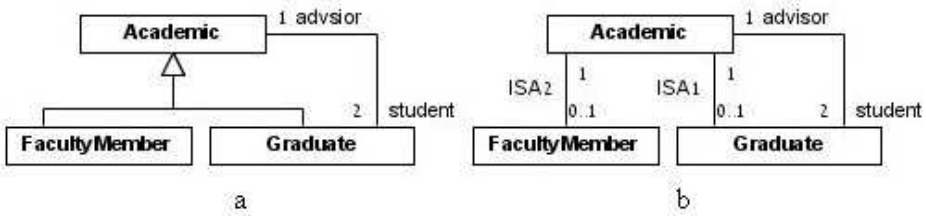


Figure 3.2: Reduced Class Diagram

Example 3.1. *Figure 3.2-b is the reduced class diagram of the class digram in Figure 3.2-a. Applying the inequalities method of [18] (step 2 in the algorithm) yields the inequalities system below. We describe the inequalities system for Figure 3.2 using the symbols ad for Academic, g for Graduate, fm for FacultyMember, as for advising-student-association and* isa_1, isa_2 *for the new associations* ISA_1 *and* ISA_2 *respectively. This system has no solution and therefore the model is not fully finitely satisfiable.*

- *1,2.* $as = 2ad, \ \ (as \geqslant 2ad, as \leqslant 2ad)$
- *3.* $as = g, \ \ (as \leqslant g, as \geqslant g)$
- *4,5.* $isa_1 = g, (\ \ isa_1 \geqslant g, isa_1 \leqslant g)$
- *6.* $isa_1 \leqslant ad$
- *7,8.* $isa_2 = fm, \ \ (isa_2 \geqslant fm, isa_2 \leqslant fm)$
- *9.* $isa_2 \leqslant ad$
- *10-15.* $ad, g, fm, as, isa_1, isa_2 > 0$

Note 3.1. *The application of our method to Figure 3.2-a. requires the addition of two associations to the reduced class diagram. The resulting inequalities system has six variables and fifteen inequalities. In comparison, the application of the [8], [9] methods to Figure 3.2-a. results a class diagram with 4 classes and 8 additional associations. The resulting inequalities system has twelve variables and twenty-six inequalities.*

Notation 1 We call the new binary associations defined in Algorithm 3.1, ISA-association and denote them by $Isa_{ij}(super : C_i[1,1],\ \ sub : C_j[0,1])$ where C_i and C_j are the superclass and the subclass, respectively. Instances of CD' classes which are ISA-related are called ISA-related objects or ISA-linked tree if it constitutes a tree.

Claim 3.3. ***[Correctness]*** *Algorithm 3.1 tests all class finiteness of a class diagram with unconstrained generalizations, with at most a single generalization set for a super class.*

Proof. Since the [18] method tests *all class finiteness*, it is sufficient to show that step 1 of the algorithm defines a reduction of the *all class finiteness* problem for CD [T]-GS to the *all class finiteness* problem of CD' without generalization sets. We have to show if and only-if mapping of the problems.

If-part:

Assume that CD' is *all class finitely satisfiable*, and let I' be a non-empty legal instance, (hence finite). We have to prove that in that case CD is also *all class finitely satisfiable*. We prove this by constructing a non-empty legal instance I for CD.

The construction of I relies on a corresponding T that we construct from objects in I' to objects in I. The mapping T is not 1:1. It maps all the objects in a tree of ISA related objects in I' (ISA-linked tree) into a single object in I. The intuition is that CD' splits a single instance object of CD into its components in its ancestor classes. For example, *olga* in Figure 3.3-a is both a *Graduate* and *Undergraduate* student (different faculties) is split into the tree rooted in *olga* in Figure 3.3-b. The instance *meir* in Figure 3.4-a is another object of the class *Graduate* alone. Therefore, it is split in Figure 3.4-b into the tree rooted in *meir* with one ISA-link to an instance of *Graduate*.

1. **Population of CD-classes - Defining class extensions for I**: The ISA links divide the objects domain into unconnected ISA-linked trees (since CD is a [T]-GS class diagram with tree structure class hierarchies). Objects of classes that are not part of any hierarchy form single object trees. An ISA-linked tree satisfies property 3.1:

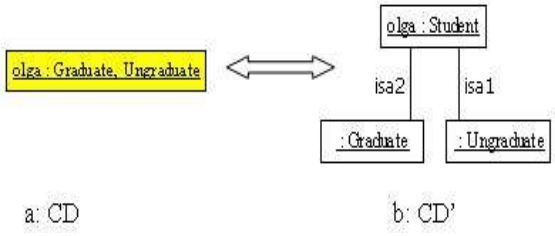


Figure 3.3: Instances of Figure 3.2-a and b

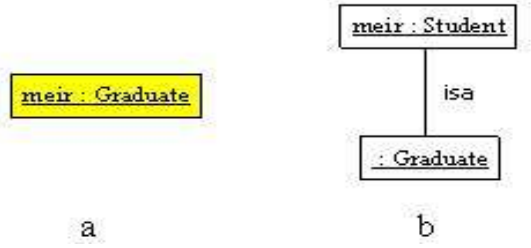


Figure 3.4: Instances of Figure 3.2-a and b

Property 3.1. *An ISA-linked tree in I' cannot include two instances of the same class. This property results from the tree structure of class hierarchies in CD. It ensures that the mapping T preserves the cardinality constraints in I.*

Now we present the population of CD-classes. Each ISA linked tree of objects in I' corresponds to a single new object in I , that instantiates all classes of the tree. For example, for the tree in Figure 3.5: insert a single new object $\overline{e}$ to the classes C_1, $C_{11}...C_{nn}$ in the tree. $\overline{e}$ is a common object for all classes in the tree.

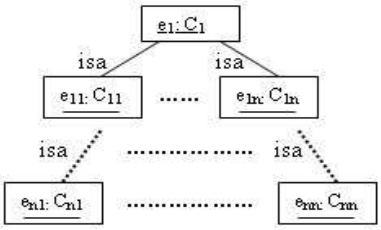


Figure 3.5: Tree of ISA links

For every object e' in a tree in I', let $T(e') = \overline{e}$. This construction defines the extension of CD classes in I. It satisfies property 3.2:

Property 3.2. $e' \in C^{I'} \Rightarrow T(e') \in C^I$

Adding links to I. A non-ISA link of an association α in I' is translated into a link of the association α between $T(e_1)$ and $T(e_2)$. For example, the link in Figure 3.6 is translated into a link in Figure 3.7.

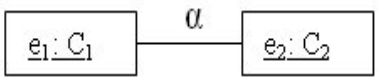


Figure 3.6: A non ISA-link

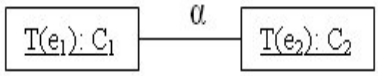


Figure 3.7: A Translated Link of Figure 3.6 in CD

This is well defined due to property 3.2.

Auxiliary Claim. I is a legal, *fully finite instance* of CD. That is:

1. I is finite and has non-empty extensions for all classes.

2. I satisfies cardinality constraints, association inheritance and class hierarchy constraints of CD.

Proof. 1. I is finite and non-empty: All classes in I are not empty and finite by property 3.2 and since all the class extensions in I' are so.

2. Hierarchy constraints: For every ISA link tree in I', an object that represents the whole tree is created, and inserted to the extensions of all classes in the tree. Therefore, every object in a sub-class belongs also to all of its super-classes.

3. Cardinality constraints: Consider an association α in CD: The cardinality constraints on α are satisfied by the α-links in I'. By the T construction, all links are mirrored in I. Therefore, for e $C^{I'}$ all of its links are preserved by $T(e)$. In order to show satisfaction of the cardinality constraints we need to observe new links of $T(e)$ in I.

Indeed, since T collapses an ISA-tree in I' into a single object in I, it is possible that $T(e)$ has more links than any of its origins (in fact it has the union of all these links). However, by Property 3.1, an ISA-tree in I' does not include two objects of the same class. Therefore, the number of the α-links of e in I' is preserved by its mirror $T(e)$ in I ($T(e)$ might have links for new associations, as it might be member of other classes, beyond C). Consequently, the cardinality constraints of α in I are satisfied.

4. Association Inheritance: An object $T(e)$ in I has all the links of its origins in I'. In particular $T(e)$ as an object in a subclass also belongs to all the super-classes in the generalization set, and has all the links of its origins in these classes. Therefore, it inherits the associations of the super-classes and satisfies their cardinality constraints.

□

Only-if-part.

Assume CD is *all class finitely satisfiable* and let I be a non-empty legal instance of CD. We construct such an instance I' for CD'. It is based on an inverse mapping T^{-1} of T from the *If-part*-direction.

1. Population of CD' classes - defining class extensions for I': An object in I that belongs to classes $C_1, ..., C_n$ is split into an n new objects of $C_1, ..., C_n$ in I'. That is, for $e \in {C_1}^I, ..., {C_n}^I$:

   (a) $T^{-1}(e) = \{e_1, ..., e_n\}$, where all e_i are new objects, and $e_i \in {C_i}^{I'}$, $1 \leqslant i \leqslant n$. e_i is denoted $T^{-1}|_{(C_i)}(e)$.

   (b) For $e \in {C_1}^I, ..., {C_n}^I$, for these classes that form a generalization set, their corresponding elements from within the $e_1, ...e_n$ are linked by the appropriate ISA links. For example, consider the tree hierarchy in Figure 3.8-a, and assume that I includes $e \in {C_1}^I, ..., {C_7}^I$. Then each of the objects set $\{e_1, e_2, e_3\}$, $\{e_2, e_4, e_5\}$, $\{e_3, e_6, e_7\}$ will be linked by the appropriate ISA links, as shown in Figure 3.8-b.

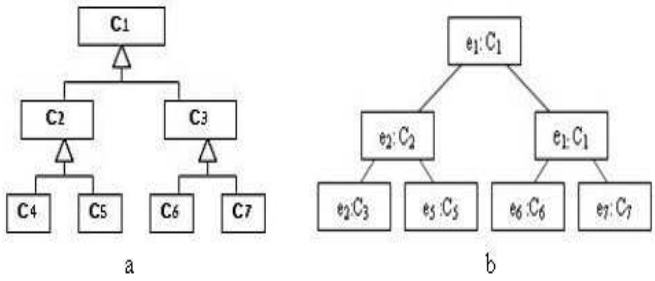


Figure 3.8: Unconstrained Generalization Sets

This construction defines the extension of CD' classes in I' and the ISA links. It satisfies property 3.3 presented below:

Property 3.3. $e \in C^I \Rightarrow T^{-1}(e) \bigcap C^{I'} \neq \emptyset$

2. Adding associations to I': A link α in I (see Figure 3.6) is translated into a link of the association α between the C_1 and C_2 translations of e_1, e_2 respectively, this is shown in Figure 3.9:

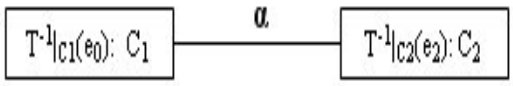


Figure 3.9: A Translated Link of Figure 3.6 in CD'

This is well defined due to property 3.3.

Auxiliary Claim. I' is a legal, fully finite instance of CD'. That is:

1. I' is finite and non-empty.
2. I' satisfies CD'. That is it all satisfies all cardinality constraints:
   (a) On ISA links
   (b) On original CD associations

Proof.

1. I' is finite and non-empty: I' is not empty since all the class extensions in I are not empty, and according to property 3.3 above, the objects of a class extension in I yield objects for the same class extension in I'. Finiteness of I' derives from the same argument.

2. ISA Cardinality Constraints: The ISA links in I' are created between objects whose corresponding classes form a generalization set. Since I satisfies class hierarchy constraints in CD, an object e in ${C_1}^I$ also belongs to its super-classes ${C_2}^I$. Therefore, $T^{-1}|_{C_1}(e)$ is ISA-related to $T^{-1}|_{C_2}(e)$ in I'. Since ISA links in I' always involve new objects the maximum cardinality of 1, in both ends of an ISA constraint is satisfied.

3. Non-ISA Cardinality Constraints: Consider the association α in CD. The cardinality constraints of α are satisfied by the α-links in I. By the T^{-1} construction, for an object $e \in {C_1}^I, ..., {C_n}^I$, its mirror objects in I' preserve all the original links. A mirror object $T^{-1}|_C(e)$ in I' has no new α-links, beyond those derived from I. Therefore, the cardinality constraints on α are satisfied in I'.

end of proof of Claim 1.

□

Claim 3.4 (Complexity). . *Algorithm 3.1 adds to the [18] method an $O(n)$ time complexity, where n is the size of the class diagram (including associations, classes and ISA constraints).*

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

3.2 Extension of Algorithm 3.1 to Class Diagrams with {[M], [A], [G]}-GS

Algorithm 3.1 also applies properly to the rest of the unconstrained structures: an unconstrained tree hierarchy, with multiple GS per super class, an unconstrained acyclic hierarchy and an unconstrained graph hierarchy ({[M], [A], [G]}-GS, respectively). The proof is the same as that of Claim 3.3 for both directions, except that we need to prove in the Auxiliary Claim *If-part* that I satisfies the cardinality constraints . For that purpose, it is sufficient to show that Property 3.1 also holds for {[M], [A], [G]}-GS. This ensures that the T construction preserves the cardinality constraints. The rest of the proof remains the same.

3.2.1 Extension of Algorithm 3.1 to Class Diagrams with [M]-GS

By definition, the ISA links divide the objects domain of I' into disjoint ISA-link trees (since CD is an [M]-GS class diagram with tree structured class hierarchies). Therefore, Property 3.1 is satisfied.

3.2.2 Extension of Algorithm 3.1 to Class Diagrams with [A]-GS

Lemma 3.5. *If ISA-related object structure in I' includes two objects from the same class, the undirected ISA association structure is cyclic.*

Proof. Let us assume that the objects a_1, $a_2 \in A^{I'}$ belong to an ISA-related object structure in I'. Let $P(a_1, a_2)$ denote the shortest (undirected) path between a_1 and a_2 in I' (there must exist such a path). Proof by induction on the shortest path $|P(a_1, a_2)|$ length.
First we show that path-lengths of 2 or 3 cannot form a basis of the induction:

- $|P(a_1, a_2)| = 2$: There are 2 objects a_1, a_2 with an ISA-link as shown in Figure 3.10-(a). Contradictory to acyclicity of the ISA association in CD'. Therefore, this path is not legal and it cannot form a basis of the induction.

– $|P(a_1, a_2)| = 3$. $P(a_1, a_2) = a_1, b_1, a_2$. Yet, for the same reason above, b_1 cannot be an object of A. Therefore, $b_1 \in B^{I'}, B \neq A$, which means that b_1 has two ISA-links to objects a_1 and a_2 as shown in Figure 3.10-(b). However, such instance violates the cardinality constraints of the ISA association. This path cannot form a basis of the induction.

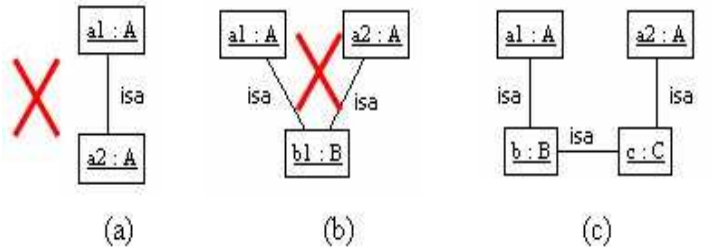


Figure 3.10: Different ISA-Paths

Basis:

$|P(a_1, a_2)| = 4$: By previous cases, $P(a_1, a_2) = a_1, b_1, c_1, a_2$, where $b \in B^{I'}$, $c \in C^{I'}$ and $A \neq B \neq C$ as shown in Figure 3.10-(c). The instance in Figure 3.10-(c) preserves the acyclicity of the ISA and the cardinality constraints. Therefore, $|P(a_1, a_2)| = 4$ constitutes the induction base. Now, we will show that for all cases of ISA associations in CD' involving A, B and C, preserve the acyclicity of ISA and the cardinality constraints, and have the path $P(a_1, a_2)$ in Figure 3.10-(c) as a legal instance, CD' is not an [A]-GS structure. Indeed, Figure 3.11 lists all these cases. We can note that the undirected ISA association structure in all cases is cyclic. Therefore, CD' is not an [A]-GS structure.

Inductive Step:

$|P(a_1, a_2)| > 4$. Let $P(a_1, a_2) = a_1, ...b_1, ...c_1, ...a_2$, *where* $b_1 \in B^{I'}$, $c_1 \in C^{I'}$ *and* $A \neq B \neq C$, as shown in Figure 3.12-(a). If such b and c do not exist, there is an ISA association cycle in CD', by the induction assumption. Since I' is a legal instance, there are, in CD', ISA association paths connecting A with B, B with C and C with A. Therefore, there is an association path connecting A, B and c, as shown in Figure 3.12-(b), implying that the

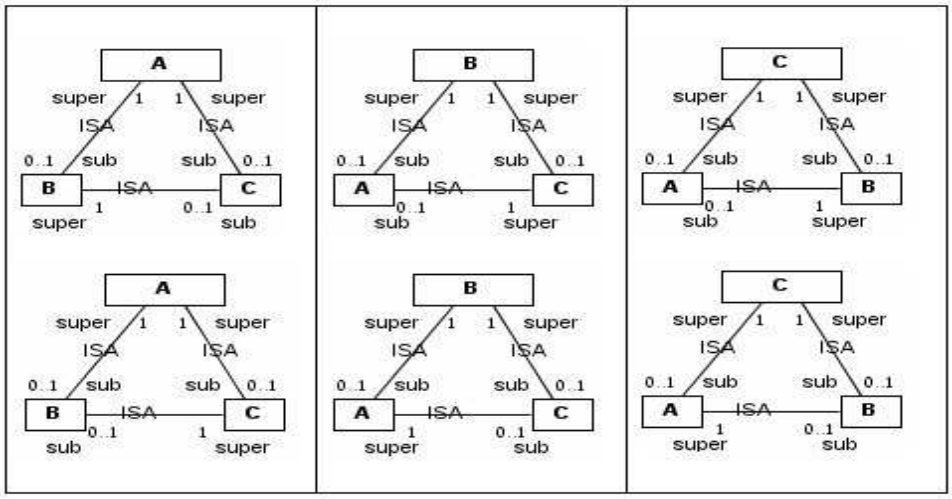


Figure 3.11: All Cases of ISA Associations for Figure 3.10-c

ISA association graph in CD' is cyclic.

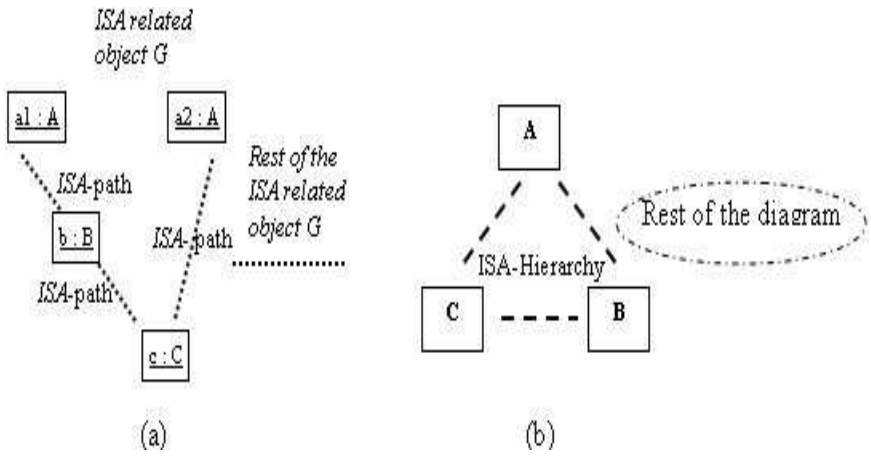


Figure 3.12: Cyclic Graph

Therefore, if CD' is [A]-GS, the objects in a connected ISA link structure in its legal instances include objects from different classes. □

3.2.3 Extension of Algorithm 3.1 to Class Diagrams with [G]-GS

Unlike the previous structures where Property 3.1 is inherently satisfied in every arbitrary legal instance I', an arbitrary legal instance I' of class diagram with a [G]-GS structure

does not always satisfy Property 3.1. For example, consider the class diagram and its transformation in Figure 3.13-a and b respectively. Figure 3.14-a is a legal instance of the reduced class diagram in Figure 3.13-b. The instance in Figure 3.14-b was obtained after applying the mapping T to the instance in Figure 3.14-a. The problem is that the instance in Figure 3.14-b does not constitute a legal instance of the class diagram in Figure 3.14-a since it violates the cardinality constraints on object o. It belongs to class A, but has 2 R-links. This problem occurs since the instance in Figure 3.14-a includes the two A instances a_1, a_2. A desired "good" instance" appears in Figure 3.20. Where a "good" instance means a legal instance that satisfies Property 3.1

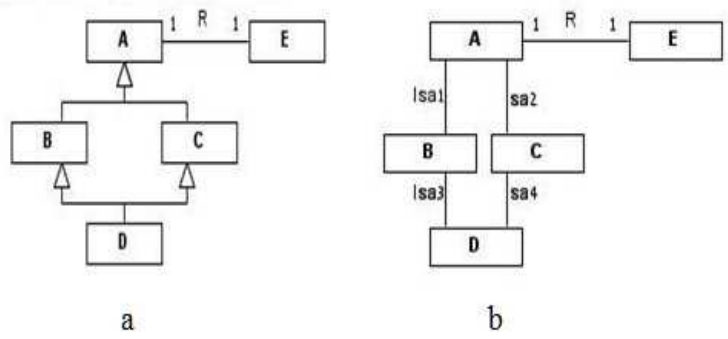


Figure 3.13: Graph Structure

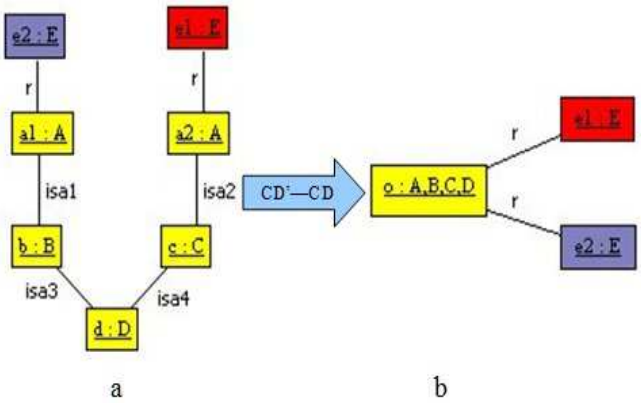


Figure 3.14: Instance of Figure 3.13 and its Mirrored in CD

The extension of Claim 3.3 to [G]-GS structures is based on the following argument:

– *Given a finitely satisfiable class diagram CD', Algorithm 3.3 constructs a "good" instance for CD'.*

It follows that every finitely satisfiable class diagram has an instance that satisfies Property 3.1 and sufficient for the correctness of Claim 3.1.

Algorithm 3.3 is a modification of the Lenzerini and Nobili proof [18]. The proof has two directions. The *If-part* direction includes a construction of a legal instance (shortly, LN-Construction) to the class diagram from a given solution to the inequalities. First, we present the LN-Construction for a non-Empty Finite Instance. For the sake of simplicity, Algorithm 3.2 is limited to binary associations only, unlike original LN-Construction. Example 3.2 demonstrates this construction. LN-Construction used the following propositions:

Proposition 3.1. *If a linear homogeneous inequality system with rational coefficients admit a positive solution, then it also admits an integer positive solution.*

Proposition 3.2. *Assume X, an integer solution for the inequality system defined for a class digram CD, where $X[T]$ denotes the value assigned by X to an unknown corresponding to the class or association T. Then, there is exist an integer solution for the inequality system which satisfies:*

– *For every relationship $R(Rn_1 : C_1[min_1, max_1], Rn_2 : C_2[min_2, max_2])$ of CD, the following condition holds: $X[R] \leqslant X[C] * X[C]$.*

Proof. If such assumption does not hold for an association $R(Rn_1 : C[min_1, max_1], Rn_2 : D[min_2, max_2])$, then multiply X to a suitably large constant, obtaining a new solution Y, that satisfies $Y[R] \leqslant Y[C] * Y[D]$. □

Algorithm 3.2. *The LN-Construction for a non-Empty Finite Instance.*

– ***Input**: A finitely satisfiable class diagram CD.*

– ***output**: A non-empty legal instance I for CD*

– ***Methods:***

1. *Obtained an integer solution X for the inequality system of CD in which for every relationship $R(Rn_1 : C[min_1, max_1], Rn_2 : D[min_2, max_2])$ of CD, the following condition holds: $X[R] \leqslant X[C] * X[D]$.*
2. *For all association $R(Rn_1 : C_1[min_1, max_1], Rn_2 : C_2[min_2, max_2]) \in CD$ do*
   (a) *insert $X[C_1]$ objects into C_1^I, $X[C_2]$ objects into C_2^I and number arbitrarily.*
   (b) *For $i = 1$ to 2 do*
      i. *For $k = 0$ to $X[R]$ - 1 do*
         – *Connect the hth object of C_i, c_{ih} to the kth link of R, r_k where, $h = k \bmod X[C_i]$.*
      ii. *Renumber R-links in such a way that links having the same set of associated objects are contiguous.*

– ***End of the Algorithm***

Example 3.2. ***Applying the LN-Construction Algorithm for the ISA Association in Figure 3.13-b***

Steps 1: First, we will present the inequality system for the ISA-association in Figure 3.12 with a possible solution:

1. ***The Inequality System for ISA Association in Figure 3.13-b:***
   – $isa_1 = b,\ isa_1 \leqslant a$
   – $isa_2 = c,\ isa_2 \leqslant a$
   – $isa_3 = d,\ isa_3 \leqslant b$
   – $isa_4 = d,\ isa_3 \leqslant c$
2. ***A Possible Solution:*** $a = 2,\ b = c = d = 1,\ , is_1 = isa_2 = isa_3 = isa_4 = 1$. *The solution satifies the restriction of step 1.*

Step 2: The Population of the ISA_1 Associations

- *Step 2.a [Population of the classes A and B + Numbering]:* $A^{I'} = x_0, y_1$, $B^{I'} = b_0$.
- *Step 2.b [Relates the class objects to the ISA-links]: Class A* $(i = 1)$*:*
  - *Step 2.b.i: The object* x_0 *is inserted into the link* $isa1_0$ *(h= 0 mod 1) as shown in Figure 3.15-(a). Step 3.a.iii.*

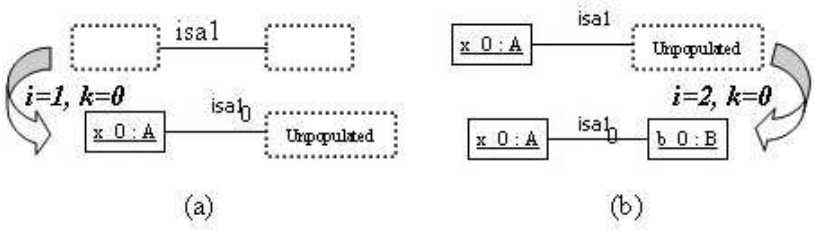


Figure 3.15: LN-Construction: ISA_1-Link Construction Processes

  - *Step 2.b.ii: This step is not executed, because there is only one instance. (Actually, this step will never be executed for links of the ISA type.)*
- *Step 2.b: Class B (i=2): The object* b_0 *(h-object of B) is inserted to the link* $isa1_0$ *(h= 0 mod 1=0) as shown in Figure 3.15-(b).*

The construction ends for this link when the object y_1 *of A remains without any connection to any other object.*

Step 2: Populating ISA_2 Association

1. *Step 2.a:* $A^{I'} = (x_1, y_0)$, $C^{I'} = c_0$
2. *Step 2. b: This building process is similar to the previous association (ISA_1). Therefore, we will not repeat the analysis. The output is presented in Figure 3.16-(a) below:*

Step 2: Populating the ISA_3 Association.

1. *Step 2.a :* $B^{I'} = b_0$, $D^{I'} = c_0$.

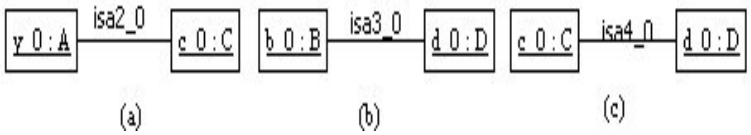


Figure 3.16: LN-Construction: ISA-Links

2. *Step 2.b: The output is presented in Figure 3.16-(b).*

Step 2: Populating ISA_4 Association.

1. *Step 2.a: $C^{I'} = c_0$, $D^{I'} = d_0$.*

2. *Step 2.b: The output is presented in Figure 3.16-(c).*

The final constructed instance is described in Figure 3.17.

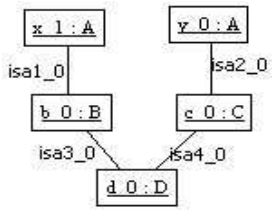


Figure 3.17: LN-Construction: Final Instance

We can see that this instance has two objects from the same class. Therefore, it does not satisfy Property 3.1. The problem stems from changing the numbering of the objects of A during the construction of the instances of the second association. As a result, the y object received an index of 0 and is connected to the zero object of C.

Algorithm 3.3 modifies the LN construction in a way that guarantees the construction of a good instance. This is due to fixing the class objects ordering (step 2.b.i in LN Algorithm) prior to the construction of the links.

Algorithm 3.3. *Modified LN-Construction for a "good" Instance.*

– ***Input**: A finitely satisfiable class diagram CD.*

- ***output***: *A non-empty "good" instance I for CD.*
- ***Methods***:
  1. *Obtained an integer solution X for the inequality system of CD in which for every relationship $R(Rn_1 : C[min_1, max_1], Rn_2 : D[min_2, max_2])$ of CD, the following condition holds: $X[R] \leqslant X[C] * X[D]$.*
  2. *For each class C in CD, define $X[C]$ instances in I' and number arbitrarily.*
  3. *For each association $R(Rn_1 : C_1[min_1, max_1], Rn_2 : C_2[min_2, max_2])$ in CD:*
     - *Populate R as in LN-Construction (Algorithm 3.2).*
- ***End of the Algorithm***

Example 3.3. *Now we will apply Algorithm 3.3 (the modified version of LN-Constricting), which performs the numbering only once, to the same example in Figure 3.13-b. Most of the analysis is similar to the analysis above, so we will only show here the end-result for each association.*
Step 1: $X[A] = 2,\ X[B] = X[C] = X[D] = 1,\ , X[ISA_1] = X[ISA_2] = X[ISA_3] = X[ISA_4] = 1$
Step 2: $A^{I'} = x_0, y_1,\ B^{I'} = b_0,\ C^{I'} = c_0,\ D^{I'} = d_0$.
Step 3 [The Population of the ISA Associations]:

1. *ISA_1 association: Figure 3.15-(b).*
2. *ISA_2 Association: Figure 3.18 below:*

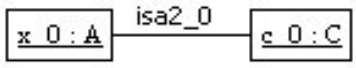


Figure 3.18: Modified LN-Construction: An ISA_2-Link Output

3. *ISA_3 Association: The output is presented in Figure 3.16-(b).*

4. ISA_4 Association: The output is presented in Figure 3.16-(c).

The final instance is shown in Figure 3.19 below.

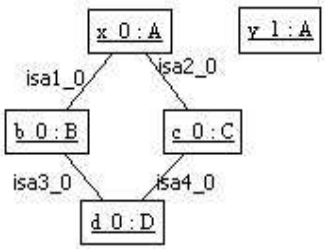


Figure 3.19: Modified LN-Construction: Final Instance

As we can see, all objects of the ISA-related object structure in Figure 3.19 have the same index. Unlike the original algorithm, preserving the original numbering ensures that two objects from the same class will not be in the same ISA-related object structure.

Proposition 3.3 (Correctness of Algorithm 3.3). *Algorithm 3.3 constructs a good instance that satisfies Property 3.1.*

Proof. Assume that CD' is all class finitely satiable, and let I' an instance for CD' constructed by Algorithm 3.3. We need to prove that I' is a good instance. That is:

1. I' is a non-empty finite instance: Algorithm 3.2 modified the LN-Construction only by fixing the class objects ordering to be done prior to the construction of the association links. It was shown in [18] that LN-Construction constructs a non empty finite instance for an arbitrary class objects ordering. Therefore, I' is a non-empty legal instance for CD'.

2. I' satisfies Property 3.1: Consider an *ISA*-association $ISA(super : C[1,1], sub : D[0,1])$ in CD'. The X solution of CD' inequality system produced by the algorithm safeties

$$X[Isa] = X[D] \ and \ X[Isa] \leqslant X[C]. \tag{3.1}$$

The LN-Construction constructs $X[ISA]$ links for the ISA-association in two separate iterations. First, the LN-Construction ISA-relates the objects of class C. In the second iteration, it ISA-relates the objects of class D. In each iteration, the LN-Construction relates the hth object of the class to the ith link of the ISA-association, where $h = \ i \ mod \ X[C]$ $(h = \ i \ mod \ X[D])$ and $0 \leqslant i \leqslant X[ISA] - 1$. However, it follows from ISA inequality 3.1, that for $0 \leqslant i \leqslant X[ISA] - 1$: $i \ mod \ X[D]) = \ i$ and $i \ mod \ X[C]) = \ i$. Hence, for the ith link of the ISA association, LN-Construction relates the ith object of C and the ith object of D. Since $X[C] >= X[D]$, it follows that the LN-Construction relates the first $X[D]$ objects of C to the objects of D, so that object c_i of C and object d_i of D are related to the ISA-link isa_i. Since all class object ordering were done prior to the construction of the links (step 2 in Algorithm 3.3), the objects in a connected ISA-link structure in I' have the same index. Consequently, they belong to different classes. Therefore, I' is a "good" instance (i.e. I' satisfies Property 3.1).

For example, Figure 3.20 presents a possible legal instance of Figure 3.13 which obtained by Algorithm 3.3. We can note that all objects in the connected ISA links have the same index.

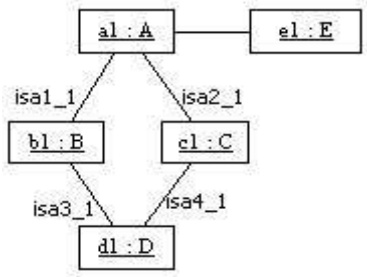


Figure 3.20: Instance of Figure 3.13

□

Claim 3.6. *If a class diagram is all class finitely satisfiable then there is least one legal instance which satisfies Property 3.1 ("good instance").*

Proof. The claim is hold due to Lenzerini and Nobili algorithm [18] and Proposition 3.3. □

Conclusion: Algorithm 3.1 tests *all class finiteness* of a class diagram with [G]-GS structure.

Chapter 4

Constrained Generalization Sets

Adding *GS*-constraints to the class diagram imposes additional requirements on its finite satisfiability problem. Figure 4.1 demonstrates a case of Infinity due to a generalization set constraint. The {*disjoint*, *incomplete*} constraint suggests that the *Academic* extension properly includes the *FacultyMember* and the *Graduate* extensions. Yet, *Academic* elements are mapped in a 1:1 manner to *Graduate* elements, implying that the sets have the same size. The only solution for proper set inclusion with equal size is that the sets are either empty or infinite. In this chapter, we extend our method to Generalization Sets with constraints, and eventually we explore the limits of our method in complicated class hierarchy structures (section 4.2). We begin with an algorithm for deciding finite satisfiability of tree structured ([T-C]-GS) class diagrams. We show that the algorithm applies also to [T-C-M]-GS and to [A-C]-GS class diagrams. Finally we explore the limits of the algorithm for the [G-C]-GS class diagrams. We show that for graph structured class hierarchies, the algorithm can handle the *overlapping* and the *incomplete GS*-constraints, but falls short for deciding finite satisfiability for the *disjoint* and the *complete GS*-constraints.

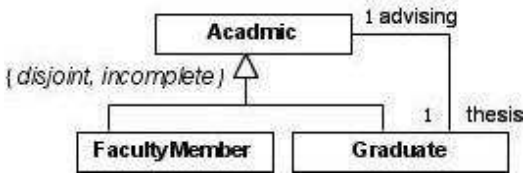


Figure 4.1: Unsatisfiability due to a generalization set constraint

4.1 Finite Reasoning Method over Class Diagram with Constrained Tree Hierarchy Structure

In order to test finite satisfiability under GS-constraints, the finite satisfiability problem of a class diagram CD, with ISA constraints, is reduced into the finite satisfiability problem of a "constrained" class diagram CD' without class hierarchy. The additional constraints on CD' preserve the constraints on the GSs of CD. The inequalities system obtained by applying the method of [18] to CD' is expanded with new inequalities that reflect the GS constraints. But, first we extend Theorem 3.1 to apply to unrestricted class diagram with constrained generalization sets.

Theorem 4.1. *If a class diagram CD with constrained generalization sets is all class finitely satisfiable, then it is fully finitely satisfiable.*

Proof

It is sufficient to prove that generalization-set constraints are closed under disjoint instance combination. The rest of the proof is the same as that of Theorem 3.1.

Lemma 4.2. *Generalization-set constraints are closed under disjoint instance combination.*

Proof. Consider a generalization set $C, C_1, ..., C_n$ with the constraint *Const*. We need to prove that all GS constraints (single or pair) are closed under a disjoint instance combination. Below, we prove that separately for each of the GS-constraints:

1. Const= *disjoint:* We need to prove that for every two classes: C_j, C_i: $C_j^I \cap C_i^I = \emptyset$. Indeed, $C_j^I \cap C_i^I = [(C_j^{I_1} \cup C_j^{I_2}) \cap (C_i^{I_1} \cup C_i^{I_2})] = (C_j^{I_1} \cap C_i^{I_1}) \cup (C_j^{I_1} \cap C_i^{I_2}) \cup (C_j^{I_2} \cap C_i^{I_1}) \cup$

$(C_j^{I_2} \cap C_i^{I_2})$. Since both I_1 and I_2 are disjoint instances and satisfy the GS-constraints:

(a) $C_j^{I_1} \cap C_i^{I_1} = \emptyset \ \wedge \ C_j^{I_2} \cap C_i^{I_2} = \emptyset$.

(b) $C_j^{I_1} \cap C_i^{I_2} = C_j^{I_2} \cap C_i^{I_1} = \emptyset$

Therefore, $C_j^I \cap C_i^I = \emptyset$.

2. Const= *overlapping:* I_1 and I_2 are disjoint instances and satisfy the overlapping constraint. Therefore, each of the instances I_1 and I_2 has at least a common object of two sub-classes or more. Uniting the two instances of I_1 and I_2 to one instance only increases the number of objects of each class. Therefore, the same classes, which shared a common object under the instances I_1 or I_2, will share under the union all the common objects from both instances I_1 and I_2. (Classes that shared common objects in one instance are not necessarily the same classes that share objects in the second instance. Therefore, unification can only add common instances to the subclasses of C).

3. Const= *incomplete*: The incomplete constraint is preserved under the union for the same reasons that we presented above with regard to the overlapping constraint: each of the instances satisfies the incomplete constraint independently. Therefore, the union operation can only insert to super-class C new objects that do not belong to its subclasses.

4. Const=*complete*: An object $c \in C^I$ is either an object of C in I_1 or an object of C in I_2. However, I_1 and I_2 satisfy the complete constraint. Therefore, the object c is also an object of a subclass of C (c may belong to more than one subclass).

□

Algorithm 4.1. *Finite Reasoning Method over Class Diagram with [T-C]-GS structures.*

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

- ***Output***: *True, if CD is all class finitely satisfiable; false otherwise.*
- ***Method***:
  1. *Class diagram reduction:*
     (a) *Steps 1.a, 1.b, 1.c from Algorithm 1.*
     (b) *For every generalization set $C, C_1, ..., C_n$ in CD, add constraint Const on its classes as follows:*
     for disjoint/overlapping constraint, Const is: "there is no/(at least one) instance of class C which is associated with more than one instance from C_1,... C_n via the ISA links";
     for complete/incomplete constraint, Const is: "all/part of the instances of class C are associated with the instances of the classes $C_1, ..., C_n$ via the ISA links".
  2. *Inequalities system construction:*
     (a) *Create the inequalities system for CD′ according to the Lenzerini and Nobili algorithm.*
     (b) *For every single constraint Const added in step 1b, extend the inequalities system, as follows:*
        i. *Const = disjoint:* $C \geqslant \sum_{j=1}^{n} C_j$
        ii. *Const = complete:* $C \leqslant \sum_{j=1}^{n} C_j$
        iii. *Const = incomplete:* $\forall j \in [1, n].C > C_j$
        iv. *Const = overlapping: No additional inequality*
     (c) *For every pair of constraints added in step 1b, extend the inequalities system, as follows:*
        i. *disjoint, incomplete:* $C > \sum_{j=1}^{n} C_j$.
        ii. *disjoint, complete:* $C = \sum_{j=1}^{n} C_j$.
        iii. *overlapping, complete:* $C < \sum_{j=1}^{n} C_j$.
        iv. *overlapping, incomplete:* $\forall j \in [1, n].C > C_j$.
  3. *Apply the Lenzerini and Nobili algorithm to CD'.*

Example 4.1. *Consider Figure 4.1. The interaction between the cardinality constraint, hierarchy, and the generalization set constraints causes a finite satisfiability problem. The application of Algorithm 4.1 yields the unsolvable inequalities system presented below (same variables as in Example 3.1). Therefore, the class diagram in Figure 4.1 is not fully finitely satisfiable.*

$$isa_1 = g,\ isa_1 \leqslant ad,\ isa_2 = fm\ ,isa_2 \leqslant ad,\ as = ad,\ ad = g\ ,\ ad > 0,\ as > 0,$$

$$g > 0, fm > 0,\ isa_1 > 0,\ isa_2 > 0,\ \text{and the added inequality } ad \geqslant g + fm$$

Claim 4.3 (Correctness). *Algorithm 4.1 tests all class finiteness of a class diagram with [T-C]-GS hierarchy structure, single and pair constraint* GS.

In order to prove the above Claim 4.3, it is sufficient to prove the auxiliary claims below. The proof of both auxiliary claims will be shown mainly for single constraints. The proof in the case of pair constraints is essentially the same with suitable extensions. Therefore, we will present the extensions for the pair constraint briefly.

Auxiliary Claim 1. A class diagram CD with [T-C]-GS is *fully finitely satisfiable* iff its reduced class digram CD' together with the associated constraints is so.

Auxiliary Claim 2. Given a class digram CD with [T-C]-GS and its reduced class digram CD', then CD' (together with the constraints) is *fully finite satisfiable* iff there exists a solution for the expanded inequalities system.

Proof of Auxiliary Claim 1

If-part.

Assume CD' is *all class finitely satisfiable*, and let I' be a non-empty legal instance. Similar to the proof of Claim 3.3, we prove the *all class finite satisfiability* of CD by the same T construction that was used in the unconstrained case. It remains to show that I is a legal instance. That is:

1. I is a legal instance of CD without the GS constraints (Claim 3.1).

2. I satisfies the generalization set constraint.

Proof of 1: The proof is analogous to that of the unconstrained case. **Proof of 2**: Let $C, C_1, ..., C_n$ be a GS in CD' and let *Const* be its constraint. We will show that all the generalization set's constraints are satisfied:

1. *Const=incompleteness.* The *incompleteness* constraint states that there is an object $c' \in C^{I'}$ (the super class) which is not ISA related to objects of $C_1, ..., C_n$ (the sub-classes). Therefore, there is an ISA-linked tree (possibly single tree) in I' which

includes c' and does not include any objects from $\{{C_1}^{I'}, ...{C_n}^{I'}\}$. The T construction collapses an ISA-tree in I' into a single object in I which instantiates all classes of the tree. Therefore, the object $T(c') \in C^I$ but does not belong to any of $\{C_1, ...C_n\}$ in I.

2. *Const=disjointness.* The *disjointness* constraint states that an object of C in I' can be ISA-related to at most one object from its sub-classes. Therefore, any ISA-linked tree in I' will include at most one object of $\{C_1, ...C_n\}$. Which means, that all extensions of $\{C_1, ...C_n\}$ in I are pairwise disjoint.

3. *Const=overlapping.* The *overlapping* constraint states that there is an object $e' \in C^{I'}$ which is ISA-related to at least two objects of its subclasses. Assume that these objects are $c'_i \in {C_i}^{I'}$ and $c'_j \in {C_j}^{I'}$. Therefore, by the T construction, an ISA-linked tree, which includes e', c'_i and c'_j will be mirrored into a single object $\overline{e}$ in of the classes C, C_i and C_j ($\overline{e}$ might be a member of other classes).

4. *Const=completeness.* The *completeness* constraint states that each object of C in I' is ISA-related to at least one object of its subclasses. Therefore, by the T construction it follows that each object of C in I also belongs to at least one sub-class of C.

Only-if-part.

Assume CD is *all class finitely satisfiable* and let I be a non-empty legal instance of CD. Similar to the unconstrained case, we apply the inverse mapping T^{-1} to yield I'. Claim 3.1 was proved in section 3.1. Therefore, it remains to show that I' satisfies the constraints in step 1b of Algorithm 4.1.

Let $\{C, C_1, ...C_n\}$ be a generalization-set with the constraint *Cons*. We should show that all the generalization set's constraints in I' are satisfied:

1. *Const=incompleteness.* By definition, there exists an object $e \in C^I$ which does not belong to any sub-class of C. By the T^{-1} construction, the mirror objects of e in I' do

not belong to $\{C_1, ...C_n\}$. Hence, $T^{-1}(e)|_C$ is not ISA related to objects of $\{C_1, ...C_n\}$ in I'.

2. *Const=complete*. An object $e \in C^I$ is also an object of at least one sub-class C_i. Therefore, its mirror object $T^{-1}|_C(e)$ is ISA related to $T^{-1}|_{C_i}(e)$. Consequently, every object $e \in C^{I'}$ is ISA related to at least one object of $\{C_1, ...C_n\}$ in I'.

3. *Const=disjointness*. All the extensions of $\{C_1, ...C_n\}$ in I are pairwise disjoint. Therefore, for an object $e \in {C_i}^I$: $T^{-1}|_C(e)$ is ISA related to $T^{-1}|_{C_i}(e)$ in I' and $T^{-1}|_{C_j}(e) = \emptyset$ for $j \neq i$.

4. *Const=overlapping*. There are two classes from $\{C_1, ...C_n\}$ in I which share one or more common objects. Let us assume that $e \in {C_i}^I \bigcap {C_j}^I$. Therefore, the mirror objects $T^{-1}|_{C_i}(e)$ and $T^{-1}|_{C_j}(e)$ will be ISA related to $T^{-1}(e)$ in I'.

Pair Constraints: The proof for the case of a pair of constraints in both directions is very similar to the proof in the case of a single constraint. In the case of a constraint-pair, two constraints are present simultaneously. Therefore, for each such pair, e.g., (disjoint, complete), we just combine the separate proofs for each of the constraints independently.

Proof of Auxiliary Claim 2

Proof. If-part. Assume CD' is *all class finitely satisfiable* and let I' be a non-empty legal instance. According to [18] the number of objects of the classes and the associations in I' constitute a solution for the [18]-inequalities system. It remains to show that this solution satisfies also the generalization set inequalities.

1. *Const=*$\{disjoint\}$: The *disjointness* constraint imposes that an object of C which can be ISA-related in I' with at most one object of its sub-classes. On the other hand, each object in a sub-class must be related to exactly one object from C. Therefore, $\left|C^{I'}\right| \geqslant \sum_{i=1}^{n} \left|{C_i}^{I'}\right|$.

2. *Const*={*complete*}: The *completeness* constraint imposes that all the objects of C are ISA-related with the objects of the sub-classes. Since an object of C can be ISA-related to more than one instance from the subclasses and every instance from the sub-classes is connected to a single instance from the super-class, therefore, $\left|C^{I'}\right| \leqslant \sum_{i=1}^{n} \left|{C_i}^{I'}\right|$.

3. *Const*={*incomplete*}: By contradiction. Assume that there is a sub-class C_i such that $\left|C^{I'}\right| = \left|{C_i}^{I'}\right|$, then all the objects of C must be ISA-related with objects of C_i, a contradiction to the incompleteness constraint.(Note, the multiplicity constraints within the ISA-association satisfy the inequality $\forall j \in [1,n]. \left|C^{I'}\right| \geqslant \left|{C_j}^{I'}\right|$). Therefore, $\forall j \in [1,n]. \left|C^{I'}\right| > \left|{C_j}^{I'}\right|$.

4. *Const*={*overlapping*}: Unlike other single constraints, the overlapping constraint does not have an inequality. However, an instant which is overlapping can produce a solution that satisfies all other constraints. For example, in the two instances in Figure 4.2.a and Figure 4.2.b we have an instance which is overlapping. However, the instance in Figure 4.2.a provides a solution to the equations of the disjoint and the complete constraints. In Figure 4.2.b., an instance which is also overlapping provides the solution for the incomplete constraint equation.

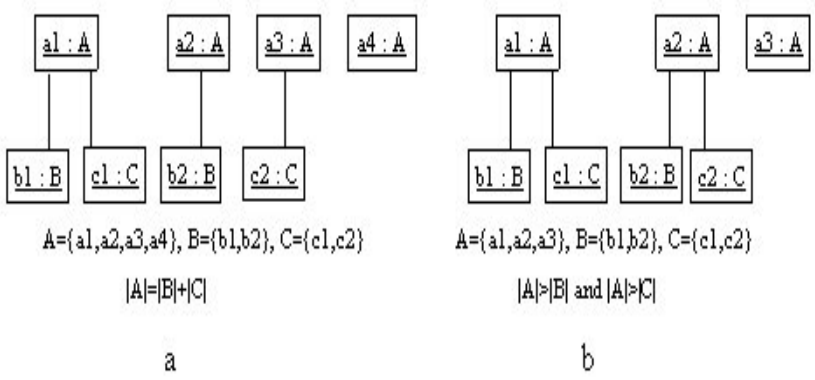


Figure 4.2: Overlapping Instances

5. *Const*=$\{disjoint, complete\}$: The combination of *disjoint* with *complete* imposes *one-to-one* and *onto* mapping between objects of C and the objects of its subclasses. Therefore, $\left|C^{I'}\right| = \sum_{i=1}^{n} \left|{C_i}^{I'}\right|$.

6. *Const*=$\{disjoint, incomplete\}$: The combination of *disjoint* with *incomplete* imposes a *one-to-one* (not *onto*) mapping between the subclass objects to objects of C. Therefore, $\left|C^{I'}\right| > \sum_{i=1}^{n} \left|{C_i}^{I'}\right|$.

7. *Const*= $\{overlapping, complete\}$: All C objects are ISA related to the objects of $C_1, ...C_n$ (complete) and there exists an object from C which is ISA related to two objects or more of the subclasses (overlapping). Therefore, $\left|C^{I'}\right| < \sum_{i=1}^{n} \left|{C_i}^{I'}\right|$.

8. *Const*= $\{overlapping, incomplete\}$: We have already shown that a necessary condition for satisfying the *incomplete* constraint is that the number of objects of superclass C be greater than the number of the objects in each subclass separately. Therefore, the only influence that the overlapping constraint can have is strengthening the inequality (e.g., the inequality of disjoint). Figure 4.2.b presents an instance which satisfies both the overlapping and incomplete constraints. This instance satisfies also the *incomplete* inequality, but not the *disjointness* inequality. Therefore, the combination of the *overlapping* constraint with the *incomplete* constraint does not require changing the *incomplete* inequality.

If-only-part. Assume that a solution for the expanded linear inequalities system defined for CD'. We need to prove that CD is all class finitely satisfiable. The proof is based on a construction of a legal instance for CD'.

The construction is given by Algorithm 4.2 below. Similar to Algorithm 3.2 in section 3.2, the algorithm is a modification of the *If-only* direction of the Lenzerini and Nobili proof. This direction includes a construction of a legal instance for the class diagram from a given solution to the inequalities. The proposed modification is correct, since any solution for the

expanded inequalities system is also a solution to the LN-inequalities system. Therefore, the *If-only* direction of the Lenzerini and Nobili proof holds.

We have already shown in Chapter 3 that the original version of the LN-Construction produces a legal instance, but not a "good" one, due to the algorithm's class objects ordering method. Likewise, using the original version of the Algorithm here produces an instance that satisfies the cardinality constraints, but not the constraints attached to GS by Algorithm 4.1. Therefore, there is a need only to change the class objects ordering in the LN-Construction, in a way that guarantees the construction of a legal instance that satisfies the associated constraints defined in Algorithm 4.1. This completes the *If-only* direction in the proof oaf Auxiliary claim 2, and thereby completes the proof of Claim 4.1, for the correctness of Algorithm 4.2.

We now present Algorithm 4.2 for the construction of a non-empty legal instance of CD' from a solution to the expanded system of inequalities. The algorithm is followed by an example and correctness claim. The Algorithm uses the same Propositions 3.1 and 3.2 from Chapter 3.

Algorithm 4.2. *The Modified LN-Construction for a Legal Instance of CD'*

- ***Input:***
  1. *A class digram CD' as produced by Algorithm 4.1.*
  2. *A solution denoted Y for the expanded linear inequalities system defined for CD'.*
- ***Output:*** *A legal instance for CD'.*
- ***Method:***
  1. *Obtain a new integer solution X for the inequality system of CD in which for every relationship $R(Rn_1 : C_1[min_1, max_1], Rn_2 : C_2[min_2, max_2])$ of CD, the following condition holds: $X[R] \leqslant X[C] * X[C]$.*
  2. *Apply the LN-Construction to the unconstrained structures in CD'.*

3. *For all* ***GS****s in* CD' *do:*

   (a) *Arrange the sub-classes in descending order of size.*

   (b) *Insert X[C] objects to super-class* C.

   (c) *For all* $ISA_i(super : C[1,1],\ sub : C_i[0,1]) \in GS$ *do*

      i. *Insert* $X[C_i]$ *objects to* C_i *and number arbitrarily.*

      ii. *Number the super-class objects as follows (based on the constraint):*

         A. *Cons= {disjoint}, {complete}, {disjoint, incomplete} or {overlapping, complete}: Move the objects of* C *which are not already* ISA*-related to the beginning. Renumber the objects, begin from 0 . If all the objects of* C *are already* ISA*-connected, then use the last numbering.*

         B. *Cons= incomplete: If* $\left|C^{I'}\right| > \sum_{i=1}^{n} \left|C_i^{I'}\right|$*: Apply rule A. Else:*

            – *If* $C_i = C_1$ *then number arbitrarily the objects of* C*. Else, no numbering will be done (which means that the objects of* C *will be numbered only one time, in the first iteration).*

         C. *Cons = {overlapping} or {overlapping, incomplete}: Apply rule B.ii.*

      iii. *Populate* ISA_i *as in LN-Construction (Algorithm 3.2).*

Example 4.2. *Demonstrating the Construction of Algorithm 4.2.*
In this example, we present the construction of Algorithm 4.2 for the class diagrams in Figure 4.3 below for each GS constraint {const}. In Chapter 3, we presented at length the LN-Construction process. We showed that LN-Construction satisfies Property 4.1.

Property 4.1. *For an association* $R(super : C[1,1],\ sub : D[0,1])$*, The LN-Construction relates the first* $\left|D^I\right|$ *objects in order of* C *to the objects of* D*, in which object* c_i *of* C *(the ith object of* C*) and object* d_i *of* D *(the ith object of D) are related to the R-link* r_i *(the ith link of R).*

In this example we only the special class ordering in GSs. The population of the ISA *associations in the GS is done as before (Algorithm 3.2). We show only the final instance.*

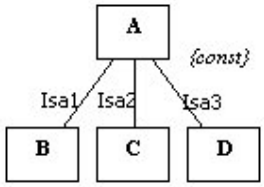


Figure 4.3: Constrained Class Diagram

The disjoint Constraint

The disjoint constraint states that an object of A can be ISA-related only to one object of the sub-classes. The object ordering rule of the algorithm is based on moving the non-ISA-related object of A to the beginning. Yet due to Property 4.1, in order to guarantee that the LN-Construction constructs a legal instance for the class diagram in Figure 4.3 that satisfies the disjoint constraint, the number of non-ISA-related objects of A must be greater or equal to the number of objects of the sub-class of the association. Indeed, I' satisfies the disjoint inequity $\left|A^{I'}\right| \geqslant \left|B^{I'}\right| + \left|C^{I'}\right| + \left|D^{I'}\right|$. Therefore, at the end of the first iteration (ISA_1 association), the number of non-ISA related objects is equal to $\left|A^{I'}\right| - \left|B^{I'}\right|$. In the next iteration, the non-ISA-related objects of A will be moved to the beginning. Due to the disjoint inequality $\left|A^{I'}\right| - \left|B^{I'}\right| \geqslant \left|C^{I'}\right| + \left|D^{I'}\right|$, the number of non-ISA-related objects of A is greater than the size of C. And at the end, the number of non-ISA-related objects will be $\left|A^{I'}\right| - \left|B^{I'}\right| - \left|C^{I'}\right| \geqslant \left|D^{I'}\right|$. Therefore, the disjoint constraint is satisfied.

1. ***Input**: The class diagram in Figure 4.3 with the disjoint constraint (i.e. an object of A can be ISA-related only to one object of the sub-classes).*

2. ***Output**: A legal instance of CD'.*

   - ***Step 1** [New Solution]: Assume that the solution: X[A]=5, X[B]= 2, X[C]=X[D]=1, $X[ISA_1]$=2 and $X[ISA_3]$=$X[ISA_3]$=1. The solution satisfies the restriction of step 1 and the disjoint inequality: $X[A] \geqslant X[B] + X[C] + X[D]$.*
   - ***Step 3.b** [Population of Class A]: $A^I = \{v, w, x, y, z\}$.*

– ***Step 3.c*** *[The Population of the ISA Associations]:*

(a) ***The population of the*** ISA_1 ***association:***

i. ***Step 3.c.i*** *[Population of Class B + Numbering]:* $B^{I'} = \{p_0, q_1\}$.

ii. ***Step 3.c.ii.A*** *[A-objects ordering]:* $A^I = \{v_0, w_1, x_2, y_3, z_4\}$ *(all A-objects are not ISA-related).*

iii. ***Step 3.c.iii*** *[output of the LN-Construction at* ISA_1*]: The output of LN-Construction at* ISA_1 *association is at as shown in Figure 4.4.*

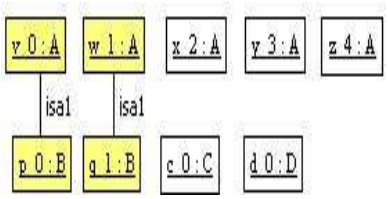


Figure 4.4: Population of ISA_1

(b) ***Step 3.c*** *[The Population of the* ISA_2 *association]:*

i. ***Step 3.c.i*** *[Population of Class C + Numbering]:* $C^{I'} = c_0$

ii. ***Step 3.c.ii.A*** *[A-objects ordering]: The* ISA_1 *iteration yields two ISA-related objects* v_0, w_1 *(see Figure 4.4), and 3 non-ISA-related objects* x_2, y_3, z_3. *The new ordering of A is as it shown in Figure 4.5.*

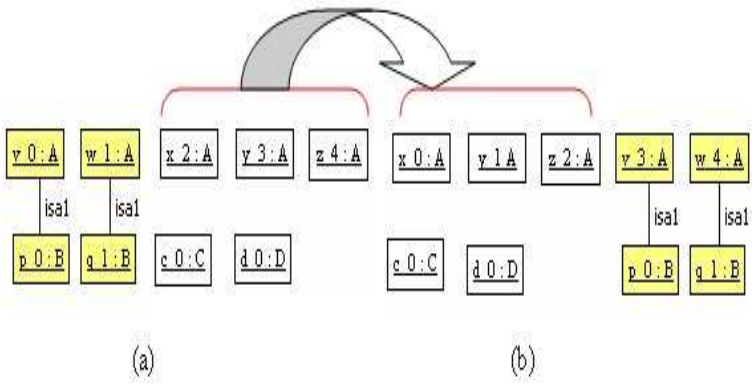


Figure 4.5: Population of ISA_1

iii. ***Step 3.c.iii*** *[Output of the LN-Construction at ISA_2]: The output of LN-Construction at ISA_2 association is as shown in Figure 4.6.*

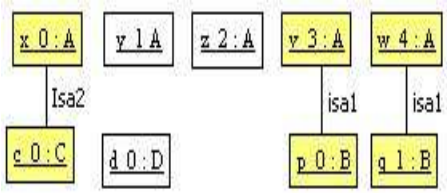


Figure 4.6: Population of ISA_2

(c) ***Step 3.c*** *[The Population of the ISA_3 Association]:*

i. ***Step 3.c.i*** *[Population of Class D + Numbering]: $D^{I'} = d_0$*

ii. ***Step 3.c.ii.A*** *[A-objects ordering]: At the end of ISA_2 iteration, there are three ISA-related objects $x_0, v_3,\ w_4$ (see Figure 4.6), and two non-ISA-related objects y_1, z_2. Hence, the new ordering of A is as shown in Figure 4.7-a.*

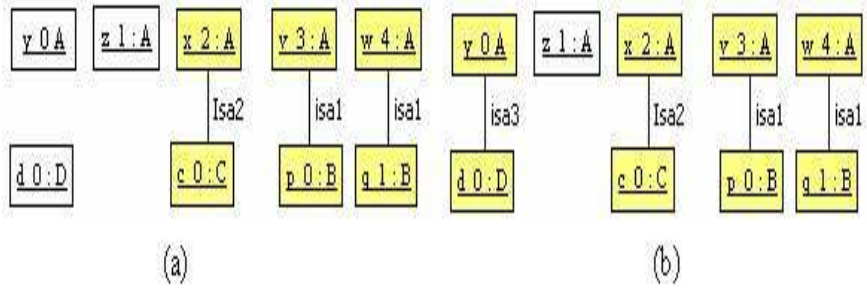


Figure 4.7: Population of ISA_1

iii. ***Step 3.c.iii*** *[Output of the LN-Construction at ISA_3]: The output of LN-Construction at ISA_2 association is as shown in Figure 4.7.*

Explanation*: Using the same consideration as in the previous ISA_2 iteration the number of the remain non-ISA-related objects is $|A^I| - |B^I| - |C^I| \geqslant |D^I|$. Hence, the reordering ensures that the LN-Construction at ISA_3 satisfies the disjointness constraints of CD' as it shown in Fig-*

ure 4.7-b.

The complete Constraint

The complete constraint states that every object of A should be ISA-related. The object ordering rule of the complete constraint is the same as the rule for the disjoint constraint. In order to guarantee that the LN-Construction constructs a legal instance for the class diagram in Figure 4.3 that satisfies the complete constraint, there must be a ISA-association of class diagram in Figure 4.3, such that the number of non-ISA related objects of A is smaller or equal to the number of the objects of the sub-class of the association. I' satisfies The complete inequality: $\left|A^{I'}\right| \leqslant \left|B^{I'}\right| + \left|C^{I'}\right| + \left|D^{I'}\right|$. *Therefore, it follows that either* $0 \leqslant \left|A^{I'}\right| - \left|B^{I'}\right| \leqslant \left|C^{I'}\right| + \left|D^{I'}\right|$ *is satisfied, or* $\left|A^{I'}\right| - \left|B^{I'}\right| - \left|C^{I'}\right| \leqslant \left|D^{I'}\right|$.

1. ***Input****: The class diagram in Figure 4.3 with the complete constraint (i.e. every object of A is ISA-related) .*

2. ***Output****: A legal instance of CD'.*

   - ***Step 1*** *[New Solution]: Assume that the solution: X[A]=3, X[B]= 2, X[C]=X[D]=1, $X[ISA_1]$=2 and $X[ISA_3]$=$X[ISA_3]$=1. The solution satisfies the restriction of step 1 and the complete inequality:* $X[A] \leqslant X[B] + X[C] + X[D]$.
   - ***Step 3.b*** *[Population of Class A]:* $A^I = \{v, w, x\}$.
   - ***Step 3.c*** *[The Population of the ISA Associations]:*

     (a) ***The Population of the*** ISA_1 ***Association****:*

     i. ***Step 3.c.i*** *[Population of Class B + Numbering]:* $B^{I'} = \{p_0, q_1\}$.

     ii. ***Step 3.c.ii.A*** *[A-objects ordering]:* $A^I = \{v_0, w_1, x_2\}$ *(all A-objects are not ISA-related).*

     iii. ***Step 3.c.iii*** *[Output of the LN-Construction at* ISA_1*]: The output of the LN-Construction at* ISA_1 *association is as shown in Figure 4.8.*

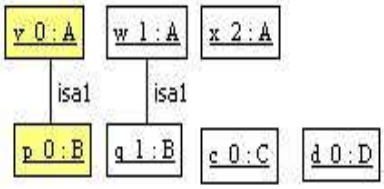


Figure 4.8: Population of ISA_1

(b) ***The Population of the*** ISA_2 ***Association:***

i. ***Step 3.c.i*** *[Population of Class* C *+ Numbering]:* $C^{I'} = \{c_0\}$.

ii. ***Step 3.c.ii.A*** *[A-objects ordering]: The* ISA_1 *iteration yields two ISA-related objects* $\{v_0, w_1\}$. *Therefore, similar considerations as in the disjointness case, the new ordering of* A *is as shown in Figure 4.9.*

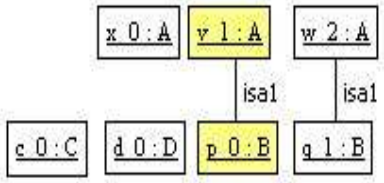


Figure 4.9: New Ordering of Class A

iii. ***Step 3.c.iii*** *[Output of the LN-Construction at* ISA_2*]: The LN-Construction relates the object* $x_0 \in A^I$ *to the object* $c_0 \in C^I$ *is as shown in Figure 4.10.*

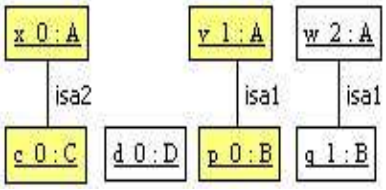


Figure 4.10: New Ordering of Class A

(c) ***The Population of the*** ISA_3 ***Association:***

i. ***Step 3.c.i*** *[Population of Class* D *+ Numbering]:* $D^{I'} = \{d_0\}$.

ii. ***Step 3.c.ii.A*** *[A-objects ordering]: At the end of the* ISA_2 *iteration, all*

object of A are ISA-related (see Figure 4.9). Hence, no numbering will be done.

iii. ***Step 3.c.iii*** *[Output of the LN-Construction at ISA_3]: The LN-Construction relates again the ISA-related object $x_0 \in A^I$ to the object $d_0 \in D^I$ as shown in Figure 4.11.*

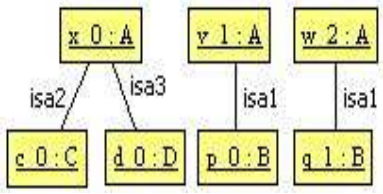


Figure 4.11: Population of the Association ISA_3

The incomplete Constraint

if $\left|A^{I'}\right| > \left|B^{I'}\right| + \left|C^{I'}\right| + \left|D^{I'}\right|$ *, then the numbering method is the same as that of the disjoint constraint. At the end of the process, there will remain* $\left|A^{I'}\right| - \left|B^{I'}\right| - \left|C^{I'}\right| - \left|D^{I'}\right| > 0$ *objects which are not ISA-related.*

In the second numbering method, the algorithm executes the ordering only once in the first iteration. Since in B has the largest size, the objects of A that will be ISA-related to classes C and D in the second iteration are objects which are already linked to B. Yet, I' satisfies the incomplete inequality: $\left|A^{I'}\right| > \left|B^{I'}\right|$*. Therefore, by the end of the last ISA-iteration there will be* $\left|A^{I'}\right| - \left|B^{I'}\right| > 0$ *non-ISA related objects.*

1. ***Input****: The class diagram in Figure 4.3 with the incomplete constraint (i.e. there is a non-ISA-related object of A).*

2. ***Output****: A legal instance of CD'.*

   - ***Step 1*** *[New Solution]: Assume that the solution: X[A]=5, X[B]= 2, X[C]=X[D]=1, $X[ISA_1]$=2 and $X[ISA_3]$=$X[ISA_3]$=1. The solution satisfies the restriction of*

step 1 and the incomplete inequalities: $X[A] > X[B]$, $X[A] > X[C]$, $X[A] > X[D]$.

- ***Step 3.b*** *[Population of Class A]:* $A^I = \{v, w, x, y, z\}$.
- ***Step 3.c*** *[The Population of the ISA Associations]:*

  (a) ***The Population of the*** ISA_1 ***Association****:*

  i. ***Step 3.c.i*** *[Population of Class B + Numbering]:* $B^{I'} = \{p_0, q_1\}$.

  ii. ***Step 3.c.ii.B-Else-part*** *[A-objects ordering]:* $A^I = \{v_0, w_1, x_2, y_3, z_4\}$ *(all A-objects are not ISA-related).*

  iii. ***Step 3.c.iii*** *[Output of the LN-Construction at* ISA_1*]: The output of LN-Construction at* ISA_1 *association is as shown in Figure 4.4.*

  (b) ***The Population of the*** ISA_2 ***Association****:*

  i. ***Step 3.b.i*** *[Population of Class C + Numbering]:* $D^{I'} = \{c_0\}$.

  ii. ***Step 3.c.ii.B-Else-part*** *[A-objects ordering]:* $C \neq B$. *Therefore, no numbering will be done.*

  iii. ***Step 3.c.iii*** *[output of LN-Construction at* ISA_2*]: The LN-Construction relates the object* $x_0 \in A^I$ *to the object* $c_0 \in C^I$ *as it shown in Figure 4.12.*

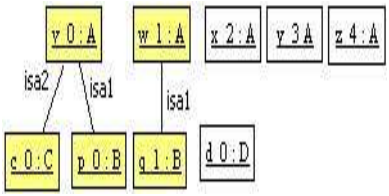


Figure 4.12: Population of the ISA_2 Association

  (c) ***The Population of the*** ISA_3 ***Association****:*

  i. ***Step 3.c.i*** *[Population of Class D + Numbering]:* $D^{I'} = \{d_0\}$.

  ii. ***Step 3.c.ii.B-Else-part*** *[A-objects ordering]:* $C \neq B$. *Therefore, no numbering will be done.*

iii. ***Step 3.c.iii*** *[Output of the LN-Construction at ISA_3]: The LN-Construction relates again the ISA-related object $x_0 \in A^I$ to the object $d_0 \in D^I$ as shown in Figure 4.13.*

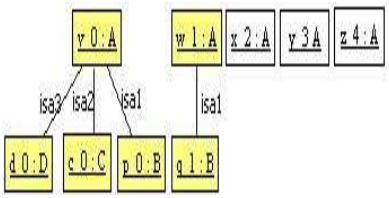


Figure 4.13: Population of the Association ISA_3

The overlapping Constraint

The overlapping constraint states there is an object of A which ISA-related to two objects of the sub-classes Similar to the incomplete case.

1. ***Input****: The class diagram in Figure 4.3 with the overlapping constraint (i.e.there is an object of A which ISA-related to two objects of the sub-classes).*

2. ***Output****: A legal instance of CD'.*

   The same example as in the incomplete case.

Proposition 4.1 (Correctness). *Let CD be a class diagram that includes binary associations and [T-C]-GS and let CD' be its transformation. Then, Algorithm 4.2 creates a legal instance I' for CD'. That is*

1. *I' is a non-empty and finite instance that satisfies the cardinality constraint.*

2. *I' satisfies the associated GS constraints defined in Algorithm 3.4.*

Proof. Assume that I' is an instance for CD' that was constructed by Algorithm 4.2. We need to prove that I' is a non-empty finite and legal instance of CD':

1. I' is a non-empty and finite instance that satisfies the cardinality constraint: The Lenzerini and Nobili proof [18] shows that for any random class ordering. Algorithm 4.2

modified the LN-construction by fixing the class objects ordering. Hence, the Lenzerini and Nobili proof holds.

2. I' satisfies the associated generalization set constraints defined in Algorithm 3.4: Let GS be a generalization set with super-class C, subclasses $\{C_1, ...C_n\}$ and a generalization set constraint *Const*. Assume, that the sub-classes are arranged in a descending order according to the number of the objects, which means that C_1 has the largest number of objects. Algorithm 4.2 builds the ISA-links iteratively, beginning from $ISA_1(super : C[1,1], sub : C_i[0,1])$ and ending with $ISA_n(super : C[1,1], sub : C_n[0,1])$. We need to prove that I' is a legal instance that satisfies each constraint *Const*. The proof is based on Property 4.1.

   (a) *Const*= *disjoint*: The *disjoint* constraint states that an object of C can be ISA related only to one object of the sub-classes. The object ordering rule for the *disjoint* constraint of the algorithm is based on moving, for each ISA iteration, the non-ISA-related objects of the super-class to the beginning. All objects of C whose order is after these non-ISA-related objects, are ISA-related. Due to property 4.1, in order to guarantee that the LN-Construction constructs a legal instance that satisfies the disjoint constraint, the number of non-ISA-related objects of C (which are ordered at the beginning) for each ISA-iteration there must be greater or equal to the number of objects of the sub-class of the ISA-association. Indeed, I' satisfies the disjoint inequality: $\left|C^{I'}\right| \geqslant \sum_{i=1}^{n} \left|C_i^{I'}\right|$. Therefore, at the end of the first ISA-iteration (ISA_1 association), the number of non-ISA related objects is equal to $\left|C^{I'}\right| - \left|{C_1}^{I'}\right|$. Due to the disjoint inequality it follows: $\left|C^{I'}\right| - \left|{C_1}^{I'}\right| \geqslant \sum_{i=2}^{n} \left|C_i^{I'}\right| \geqslant \left|{C_2}^{I'}\right|$. Therefore, in the second ISA-association, the number of non-ISA-related objects is greater than the size of C_2 (or, greater or equal for $n = 2$). Hence, in the last ISA-iteration, the number of non-ISA-related objects is equal to $\left|C^{I'}\right| - \sum_{i=1}^{n-1} \left|C_i^{I'}\right| \geqslant \left|{C_n}^{I'}\right|$.

(b) *Const=complete*: The *complete* constraint states that every object of C should be ISA-related. The object ordering rule of the complete constraint is the same as the rule for the disjoint constraint. In order to guarantee that the LN-Construction constructs a legal instance that satisfies the complete constraint, there must be an ISA-association in GS, such that the number of non-ISA related objects of C is smaller or equal to the number of the objects of the sub-class of the ISA-association. Indeed, due to the {*complete*} inequality $\left|C^{I'}\right| \leqslant \sum_{i=1}^{n} \left|C_i^{I'}\right|$, it follows that one of the following inequalities must exist: $0 \leqslant \left|C^{I'}\right| - \left|{C_1}^{I'}\right| \leqslant \left|{C_2}^{I'}\right|$, $0 \leqslant \left|C^{I'}\right| - \left|{C_1}^{I'}\right| - \left|{C_2}^{I'}\right| \leqslant \left|{C_3}^{I'}\right|$, ..., $0 \leqslant \left|C^{I'}\right| - \sum_{i=1}^{n-1} \left|{C_i}^{I'}\right| \leqslant \left|{C_n}^{I'}\right|$.

(c) *Const=incomplete*: The *incomplete* constraint states that there is at least one non-ISA-related object of C. The algorithm enables two possibilities of the ordering rule. For $\left|C^{I'}\right| > \sum_{i=1}^{n} \left|C_i^{I'}\right|$, the ordering rule is the same as in the disjoint constraint. Yet, in the last ISA-iteration, the number of non-ISA-related objects is equal to $\left|C^{I'}\right| - \sum_{i=1}^{n-1} \left|C_i^{I'}\right| > \left|{C_n}^{I'}\right|$. Hence, at the end of the last ISA-iteration, there will be $\left|C^{I'}\right| - \sum_{i=1}^{n} \left|C_i^{I'}\right| > 0$ non-ISA-related object. The second ordering rule executes the ordering only once in the first iteration. Since C_1 has the largest size, the objects of C that will be ISA-related to the remaining subclass objects are objects which are already linked to C_1. Since I' satisfies the *incomplete* inequality $\left|C^{I'}\right| > \left|C_1^{I'}\right|$. Therefore, by the end of the last ISA-iteration there will be $\left|C^{I'}\right| - \left|C_1^{I'}\right| > 0$ non-ISA related objects.

(d) *Const=overlapping*: The *overlapping* constraint states that there is an object of C which is ISA-related to two objects of the sub-classes. Indeed, the objects of C preserve their order in all ISA-iterations. Since C_1 has the most largest number of objects. Therefore, all sub-classes will be connected via the ISA-links to objects from C which are already connected to objects of C_1.

(e) *Const=* {*disjoint,incomplete*}: The object ordering rule for the {*disjoint,incomplete*}

constraint is the same as in disjoint constraint. I' satisfies the {*disjoint,incomplete*}-inequality: $\left|C^{I'}\right| > \sum_{i=1}^{n} \left|C_i^{I'}\right|$. For the disjoint constraint, the same argument as in the *disjoint* constraint. Hence, by the end of the last ISA-iteration, there will be $\left|C^{I'}\right| - \sum_{i=1}^{n} \left|C_i^{I'}\right| > 0$ non-ISA-related object.

(f) *Const*= {*disjoin,complete*}: The object ordering rule for the {*disjoin,complete*} constraint is the same as in disjoint constraint. I' satisfies the {*disjoin,complete*} inequality: $\left|C^{I'}\right| = \sum_{i=1}^{n} \left|C_i^{I'}\right|$. Hence, by the end of the last ISA-iteration, the number of non-ISA-related objects is equal to $\left|C^{I'}\right| - \sum_{i=1}^{n} \left|C_i^{I'}\right| = 0$.

(g) *Const*= {*overlapping, complete*}: The object ordering rule for the {*overlapping, complete*} constraint is the same as the one for the disjoint constraint. I' satisfies the {*overlapping,complete*} inequality $\left|C^{I'}\right| < \sum_{i=1}^{n} \left|C_i^{I'}\right|$. Therefore, , similar to the complete case, there exits an ISA-association ISA_k in $\boldsymbol{GS}$ which satisfies $(0 \leqslant |C^{I'}| - \sum_{i=1}^{k-1} |C_i^{I'}| < \left|C_k^{I'}\right|)$. Assume $m = (\left|C^{I'}\right| - \sum_{i=1}^{k-1} \left|C_i^{I'}\right|)$, Yet, due to property 4.1, the LN-Construction relates the first $\left|{C_k}^{I'}\right|$ objects of C to the objects of C_k. Since, the number of the these non-ISA-related objects is smaller than $\left|C_k^{I'}\right|$, then LN-Construction relates the already ISA-related objects of C to the object of C_k (since their orders appears after the first m non-ISA related objects of C).

□

□

Claim 4.4 (Complexity). *Method 2 adds to the Lenzerini and Nobily method an $O(n)$ time complexity, where n is the size of the class diagram (including associations, classes, class hierarchy constraints and GSC).*

Proof. The additional work involves the class diagram reduction, which creates a class diagram with the same set of classes, one additional association that replaces every class

hierarchy constraint and one additional inequality for every generalization set constraint. Since there is a linear additional work per generalization set, the overall additional work is linear to the size of the class diagram. □

Result 1: The inequalities that are used in Algorithm 4.1 for satisfying the single *GS*-constraints are not mutually exclusive. Indeed, there are solutions for the inequalities system that can imply finite satisfiability for several constraints. For example, a solution that yields equality in a *disjoint* inequality might imply that the *disjoint* constraint can be replaced by a *complete* constraint, without affecting finite satisfiability, and vice versa. Step (2.c) handles pairs of *GS*- constraints that result from combinations of *disjoint / overlapping* with *complete / incomplete*. The single constraint inequalities are tightened so to meet the combined constraints.

Result 2.The inequalities of the pair *GS*-constraints are not exclusive. The first and second inequalities imply, each, the last. Therefore, finite satisfiability with the pair constraints {*disjoint, incomplete*}/ {*disjoint, complete*} implies finite satisfiability with the {*overlapping, incomplete*} constraints. This observation leads to the following conclusion:

Conclusion: If a tree structured class diagram CD is fully finitely satisfiable, then a class diagram CD' which is obtained from CD by replacing pairs of GC-constraints {*disjoint, incomplete*}/ {*disjoint, complete*} with {*overlapping, incomplete*} is also fully finitely satisfiable.

4.2 Extension of Algorithm 4.1 to Class Diagrams with [T-C-M]-GS Hierarchy Structure

Lemma 4.5. *Algorithm 2 tests for all class finiteness of a class diagram with [T-C-M]-GS Hierarchy Structure.*

Proof. The basic idea of the proof is the same as that of Claim 4.3. We prove it by using

the same mapping T and its inverse T^{-1}. Therefore, we will not repeat the similar part. It remains to prove Auxiliary Claims 1 and 2

Auxiliary Claim 1. *If-part*: It is sufficient to prove that I satisfies the generalization set constraints. Let C be a super-class with the n-generalization sets $\{G_1, ..., G_n\}$ and the associated constraints $Const_i$ for each G_i. Therefore, in each variant of $\{Const_1, ..., Const_n\}$, there is a tree of ISA-linked objects, M in I' that satisfies these constraints. Assume that e is the translation of M in I. From Proposition 4.3 it is known that I satisfies the generalization set constraints for $n = 1$. Therefore, by looking at the restriction $M|_{G_i}$ we can conclude that e satisfies the constraint $Const_i$ of G_i. Therefore, I satisfies n-generalization sets $\{G_1, ..., G_n\}$-constraints. *Only-if-part*: Similar to the proof of Proposition 4.3.

Auxiliary Claim 2. *If-part*: Given a multiple constrained generalization sets, $\mathcal{M}$. All the constraints within $\mathcal{M}$ are satisfied simultaneously. Therefore, according to the proof in the private case, the size of the classes for each single generalization set within $\mathcal{M}$ satisfies the inequality of its constraints, together all the inequalities derived from the constraints within $\mathcal{M}$ are satisfied. *Only-if-part*: All the inequalities derived from the constraints within $\mathcal{M}$ are satisfied simultaneously. Therefore, we can apply iteratively the rules stated in *Only-if-part*-proof for each generalization set. □

Extension of Algorithm 4.1 to Class Diagrams with [A-C]-GS Hierarchy

Lemma 4.6. *Algorithm 4.1 tests all class finiteness of class diagram with [A-C]-GS Hierarchy Structure.*

Proof. The proof is similar to the previous proofs, and so we omit most of the details. We prove that I satisfies the generalization set constraints. Let C be a super class with the n-generalization sets $\{G_1, ..., G_n\}$ and the associated constraints $Const_i$ for each G_i. Let us assume that there are classes form $\{G_1, ..., G_n\}$ which are involved on another generalization sets. Therefore, in each variant of $\{Const_1, ..., Const_n\}$, there is an ***acyclic*** graph M of ISA-linked objects in I', that satisfies these constraints, assume that e is the translation

of M in I. The instance e instantiates all the classes of the restriction $M|_{G_i}$. Since $M|_{G_i}$ satisfies the constraint $Const_i$, it follows from the proof of the private case ([C-T]), that e satisfies the constraint $Const_i$. The rest of the proof is essentially the same and is therefore omitted. □

Extension to Class Diagrams with [G-C]-GS Hierarchy Structure - Exploring The Limits of the Suggested Method

Our method Algorithm 4.1 extends properly to the {[T-C-M], [A-C]}-GS hierarchy structures, but it does not extend to the full case of [G-C]-GS hierarchies. and explain why the method of Algorithm 4.1 cannot handle these cases.

Presence of a *incomplete* GS-Constraints in a [G-C]-GS class diagram: To prove the correctness of our method in this case, we need to show that if a CD' with an incomplete constraint is all class finiteness, then there is an instance I‘ that satisfies both the *incomplete* constraint and Property 3.1 (i.e., that all ISA related objects in I‘ do not include two objects from the same class).

Proof.

We present here an algorithm that builds an instance I‘ of CD‘ that satisfies both Property 3.1 and the *incomplete* constraint. The algorithm builds I‘ according to the following steps:

1. **Unconstrained generalization sets**: For the parts of CD‘ that contain unconstrained generalization sets, apply the algorithm from section 3.2. We have already proved that such algorithm satisfy Property 3.1. (All instances have the same index. Therefore, it is not possible that two instances of the same class be included in ISA-related objects).

2. **Constrained generalization sets with *incomplete***:Apply the construction algorithm from Section 2.2 to the parts of CD‘ that contain GS with the incomplete

constraint. This algorithm, as we have shown before, builds an instance that satisfies the incomplete constraint. At the same time, the construction method ensures that the instance also satisfies Property 3.1. That is so because the numbering of the instances is done only once at the beginning of the algorithm. Therefore, all ISA-related objects contain instances that have the same index.

The end-product is an instance that satisfies both the incomplete constraint and Property 3.1.

Presence of a *disjoint* GS-Constraint in a [G-C]-GS class diagram: Consider the class diagram in Figure 4.14-a. The *disjoint* constraint imposed on the generalization set $\{A,\ B,\ C, D\}$ implies that in every instance, the extension of E properly includes the extension of D. But, object members of class E are mapped in a 1:1 manner to members of D, implying that the sets have the same size. The only solution for proper set inclusion with equal size is that the sets are either empty or infinite. Therefore, the diagram is not fully finitely satisfiable.

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

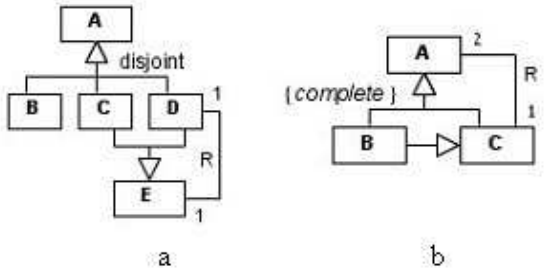


Figure 4.14: Constrained Graph Hierarchy

1. **The Inequalities System**:

(a) **Fist Generalization Set**:

- 1,2. $isa_1 = b,\ isa_1 \leqslant a$
- 3,4. $isa_2 = c,\ isa_2 \leqslant a$
- 5,6. $isa_3 = d,\ isa_3 \leqslant a$
- 7 [the disjointness constraint]: $a \geqslant b + c + d$

(b) **Second Generalization Set**:

- 7,8. $isa_4 = c,\ isa_4 \leqslant e$
- 9,10. $isa_5 = d,\ isa_5 \leqslant e$
- 7,8. $r = d,\ r = e$

2. **A Possible Solution**: $a = 3,\ b = c = d = e = 1,\ , is_1 = isa_2 = isa_3 = isa_4 = isa_5 = 1,\ and\ r = 1$

1. **The Inequalities System**:

- 1,2. $isa_1 = b,\ isa_1 \leqslant a$
- 3,4. $isa_2 = c,\ isa_2 \leqslant a$
- 5. [the completeness constraint]: $a \leqslant b + c + d$
- 6,7. $isa_3 = b,\ isa_3 \leqslant c$
- 7,8. $r = 2c,\ r = a$

2. **A Possible Solution**: $a = 2,\ b = c = 1,\ , is_1 = isa_2 = isa_3 = 1,\ and\ r = 2$

The reason for the failure of Algorithm 2 to detect that the diagram in Figure 4.14-a is not fully finitely satisfiable is lies in the projection of the *disjoint* constraint from one generalization set to the other. The implied *disjoint* constraint on the lower generalization set is not recorded in the inequalities system.

Presence of a *complete* GS-Constraint in a [G-C]-GS class diagram: Consider the class diagram in Figure 4.14-b. The *complete* constraint states that the union of the

extensions of classes B and C is the extension of class A. Yet, B is a subclass of C, implying that the extensions of C and A are equal. On the other hand, the elements of class C are mapped in a $1:2$ manner to those of class A. The only solution for having a $1:2$, *onto* mapping from a set to itself is if the set either empty or infinite. Therefore, the class diagram is not fully finitely satisfiable.

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

1. **The Inequalities System of Figure 4.14-b**:

   (a) $isa_1 = b,\ \ isa_1 \leqslant a,\ isa_2 = c,\ \ isa_2 \leqslant a,\ isa_3 = b,\ \ isa_3 \leqslant c,\ r = 2c,\ \ r = a$

   (b) The completeness inequality: $a \leqslant b + c$

2. **A Possible Solution**: $a = 2,\ \ b = c = 1,\ \ , isa_1 = isa_2 = isa_3 = 1,\ \ and\ \ r = 2$

The reason for the failure of Algorithm 2 to detect that the diagram in Figure 4.14-b is not fully finitely satisfiable lies in the projection of the $\{B,\ C\}$ generalization set on the constraints imposed on the other generalization set. The implied constraint for the $\{A,\ B,\ C\}$ generalization set is *complete, overlapping*. The addition of this constraint yields an unsolvable inequalities system.

4.3 Conclusion

Algorithm 4.1 extends properly to the {[T-C-M], [A-C]}-GS hierarchy structures, but it does not extend to the full case of [G-C]-GS hierarchies. The single GS-constraints *incomplete* and *overlapping* cause no problems. But the presence of the *disjoint* or the *complete* constraints within cyclic class hierarchies fails the algorithm

Chapter 5

Finite Reasoning Tool

5.1 Implementation

In this chapter, we present an experimental tool, which implements part of the linear programming based methods that we have developed in this thesis for reasoning about finite satisfiability in UML class diagrams. For this reasoning process, we make use of several technologies and tools in three distinct steps: **Input preprocessing**, **creating the inequalities system** and **solving the inequalities system**. (Figure 5.1 below illustrates the finite reasoning process.) The experimental tool that we developed focuses on the second step: creating the inequalities system. For the first and third steps, we utilize existing technologies and tools. The selected language for the implementation is Java.

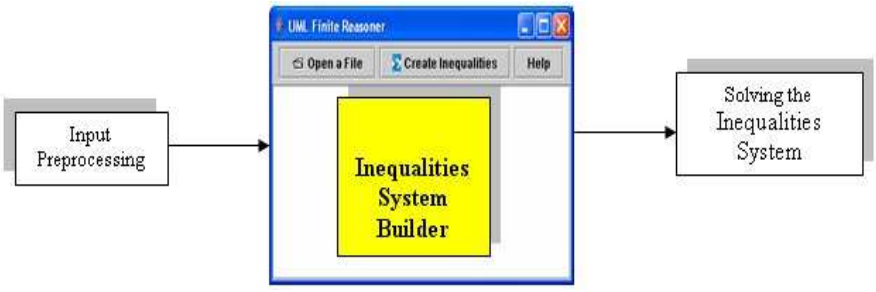
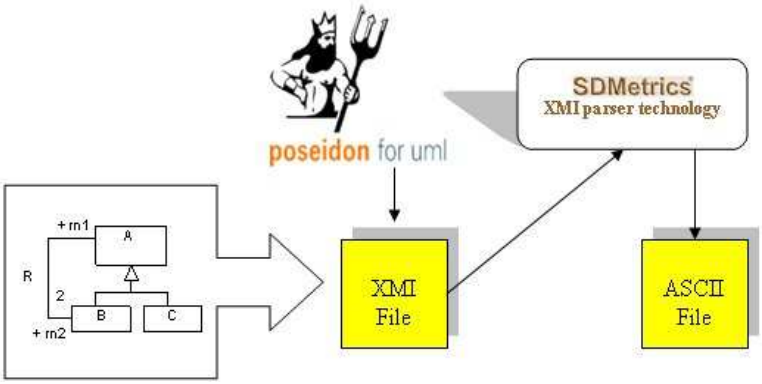


Figure 5.1: The Implementation Structure

1. Input Preprocessing In order to develop a stand-alone tool, which is platform independent and at the same time compatible with current CASE tools, we use as input the XMI standard representations of UML models. However, different versions of the XMI exist and some tools use proprietary extensions of the UML Metamodel. We solve this problem by using *SDMetrics* tool [58]. SDMetrics analyzes the structural properties of UML designs and it is embedded with an *' XMI parser technology*. SDMetrics parser imports the XMI file and extracts the information into an ASCII file which includes the "row" UML model information. The ASCII file includes a representation of UML design in a table format showing the model's elements in rows and the model's element attributes in columns. Therefore, the input for representing class diagrams in our implementation is a table format text file. Figure 5.1 illustrates the input preprocess. The Poseidon for UML CASE tool is used to build the UML models and to generate the input XMI file [57].



2. The Experimental Tool: Creating the Inequalities System Our experimental tool takes as input the table format representation of the UML class diagram from step 1 and creates an inequalities system according to the linear programming based methods, which we have developed in this thesis. The experimental tool includes two primary parts:

1. **Build Data Structure Part:** This module is responsible for creating the data structure for the class diagram's element types.

2. **Creating the Inequalities System Part:** This part is responsible for creating the inequalities system of the model. This part includes three different modules:

   (a) **Original Association Inequalities Builder:** This part is responsible for creating the inequalities of the original associations.

   (b) **ISA Association Cardinality Inequalities Builder:** This part is responsible for creating the inequalities of the ISA associations.

   (c) **Generalization Set Constraint Inequalities Builder:** This part is responsible for creating the inequalities of the generalization set constraints.

The final output is an inequalities system that represents the model. Figure 5.3 below shows a partial output of the inequalities of the class diagram from Figure 5.2.

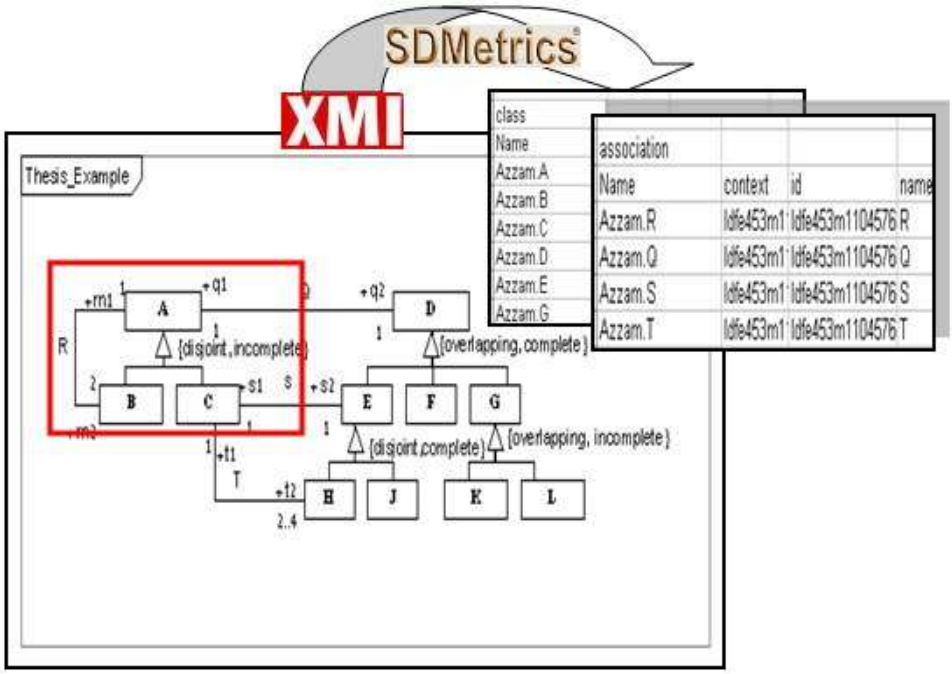


Figure 5.2: Class Diagram with a Partial Table Format Representation

3. Solving the Inequalities System: The output of the inequalities system will be moved to another case tool which decides whether the inequalities system has a solution or

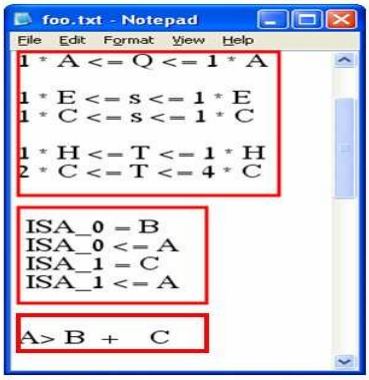


Figure 5.3: Partial Output of the Inequalities

not. For that purpose, we use the MATLAB 7.0 CASE tool. Based on the answer from MATLAB it is determined whether the model is all class finitely satisfiable or not. Below is the inequalities system that we get from our prototype tool for the diagram in Figure 5.3. The output that we get from applying MATLAB to this inequalities system is that it has no answer. **Therefore, the diagram in Figure 3 is not all class finitely satisfiable.**

5.1.1 Conclusions

The prototype tool that we present here succeeds in implementing the methods that we developed in this thesis. This tool highlights the simplicity and efficiency of our methods. It also shows the great advantage which is gained by providing an automated reasoning support for detecting inconsistencies. Such reasoning support helps greatly in improving design quality and supporting application construction needs.

Chapter 6

Conclusions and Future work

In this thesis, we developed methods for detecting and addressing problems of finite satisfiability in UML class diagrams in a way that is simple and efficient and that provides the foundations for expanding UML case tools to address these finite satisfiability problems. Furthermore, this thesis made a number of significant contributions to improving our conceptual understanding of the problems and semantics of reasoning in class diagrams. It also lays the foundations for future work in which the methods developed here may be expanded to address additional problems of class diagrams.

The model driven architecture approach emphasizes the central role that models play towards achieving reliable software. Current case tools do not support reasoning tasks about class diagrams and thus permit the construction of erroneous models. There is an urgent need for strengthening Computer Aided Software Engineering (CASE) tools with features at the level of modern integrated development environments. Such tools need to detect errors in models, identify the source of errors, reveal redundancies, and possibly suggest design improvements. These tasks require reasoning capabilities since they depend on the meaning of the constraints that occur in a diagram.

To address some of these challenges, we have, in this thesis, developed linear programming based methods for reasoning about finite satisfiability for the combination of cardinality

constraints, class hierarchy constraints, and generalization sets constraints. The extension is based on a preprocessing reduction of full finite satisfiability of a given class diagram, to the full finite satisfiability of a restricted class diagrams handled by the Lenzerini and Nobili method [18]. The method introduces only a linear number of new associations and inequalities (linear in the diagram size) and requires only a linear inequalities solver. It improves over previous extensions of the Lenzerini and Nobili method that require the addition of an exponential number of new entities to the original diagram [8]. The Prototype that we have developed and presented in Chapter 5 illustrates the efficiency and simplicity of extending UML CASE tool with such reasoning methods. It also shows the great advantage which is gained by providing an automated reasoning support for detecting inconsistencies.

However, the generalization set constraints present the hardest requirement. Their occurrences require the addition of new inequalities, and restrict the method scope to class hierarchies whose undirected graph structure is acyclic. We have shown that for graph structured class hierarchies, the algorithm can handle the overlapping and the incomplete GS-constraints, but falls short for deciding finite satisfiability for the disjoint and the complete GS-constraints. The reason is that in general graph structured class hierarchies these GS constraints have an implicit global effect on other generalization sets in a cycle. Our examination of the limits of our method in the presence of cyclic class hierarchy can contribute towards the development of a concrete reasoning method regarding cyclic class hierarchy.

In addition to developing the concrete methods described above, the thesis makes an important contribution to the conceptual understanding and the semantics of reasoning problems in class diagrams. In the thesis, we have classified the major consistency and finite satisfiability problems that require reasoning on UML class diagrams, surveyed the existing approaches and results and suggested additional notions for both satisfiability and consistency notions. These additional notions contribute to a more accurate and comprehensive understanding of reasoning problems. The survey of reasoning techniques distinguishes between transformation-based approaches and concrete methods. In particular, we surveyed

reasoning methods about emptiness and finite satisfiability problems in class diagrams.

In terms of semantics, we discussed the semantics of the overlapping and disjoint constraints and suggested the local and global interpretations for each of these constraints. Although we adopted the local interpretation for both constraints, we believe that the global interpretation is also valid.

There are a number of additional interesting issues related to reasoning about class diagrams that fall beyond the scope of this thesis, but whose resolution may build on the work that we did here. It is our hope to address a number of these issues in our future research.

1. One possible direction for building on the research done in this thesis is extending our methods to handle additional elements of class diagrams, such as:

   (a) Generalization sets with global disjointness and global overlapping constraints.

   (b) N-ary associations with both interpretation of cardinality constraints, association classes and association hierarchy constraints, association classes and association hierarchy constraints.

   (c) Our method does not apply to cyclic class hierarchies. Therefore, it seems reasonable to combine it with the Calvanese and Lenzerini [8] method. That is, use the first method for the major part of a class diagram, and apply the latter more expensive method to the cyclic class hierarchies in the diagram. This method integration relies on the assumption that cyclic class hierarchies do not occur frequently in class diagrams.

   (d) The thesis does not discuss the qualifier constraint. Building on the research and methods developed in this thesis, it would be possible to examine this constraint, highlight the finiteness problems that it can cause and develop a reasoning method to deal with these problems.

2. Another direction of possible future stream of research involves the possibility of expanding our methods with heuristics for detecting and repairing finite satisfiability problems, following the ideas of [16].

3. In order to allow automatic reasoning which is based on the linear programming methods from a Case-Tool, it would be possible to integrate our implementation as add-in in existing case tool, such that the reasoning process can be executed in a suitably integrated way.

References

[1] Balaban, M. and Shoval, P. MEER – An EER Model Enhanced with Structure Methods. Information Systems, 27, 245-275, (2002)

[2] M.Balaban, P.Shoval: Enforcing Cardinality Constraints in the ER Model with Integrity Methods. Advanced Topics in Database Research, Vol. 1 2002: 1-16

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

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

[5] Guizzardi, G., Wagner G., Guarino, N., van Sinderen, M.: An Ontologically well-Founded Profile for UML Conceptual Models. 16th International Conference on Advanced Information Systems Engineering (CAiSE), Latvia, (2004)

[6] Baclawski, K., Kokar, M., Smith, J., Letkowski, J.: Consistency Checking of Ontologies Expressed in UML. International Conference on Formal Ontologies in Information Systems, (2001)

[7] Cranefield, S., Hausteiny, S.: Purvis, M.: UML-Based Ontology Modelling for Software Agents. Proc. of Ontologies in Agent Systems Workshop, Montreal, (2001)

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

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

Martin, R.C. (2003). UML for Java Programmers, Prentice Hall

[10] Liang. P, Formalization of Static and Dynamic UML Using Algebraic. Master's thesis, University of Brussel (2001)

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

[12] Hartmann, S. On the Consistency of Int-cardinality Constraints. In Proceedings of the 17th International Conference on Conceptual Modeling, LNCS 1507, 150-163, (1998).

[13] Hartmann, S. On Interactions of Cardinality Constraints, Key, and Functional Dependencies. In Proceedings of the First International Symposium on Foundations of Information and Knowledge Systems, LNCS 1762, 136-155, (2000).

[14] Hartman, S, On the Implication Problem for Cardinality Constraints and Functional dependencies. Ann.Math.Artificial Intelligence, (2001)

[15] Engel, K. and Hartmann, S. Minimal Sample Databases for Global Cardinality Constraints. In Proceedings of the Second International Symposium on Foundations of Information and Knowledge Systems, LNCS 2284, 268-288, (2002).

[16] Hartman, S, Coping with inconsistent constraint speciation., LNCS, Vol 2224, (2001)

[17] Hartman, S, Soft Constraints and Heuristic Constraint Correction in Entity- Relationship Modeling, LNCC, Vol. 2582, (2001)

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

[19] Kozlenkov, A., Zisman, A.: Discovering Recording, and Handling Inconsistencies in Software Specifications. Int. J. of Computer and Information Science 5(2), (2004)

[20] Lange, C., Chaudron, M., Muskens. J.: In Practice: UML Software Architecture and Design Description. IEEE Software, vol. 23, no. 2, (2006)

[21] Beckert, B., Keller, U. and. Schmitt, P.H. Translating the Object Constraint Language into first-order predicate logic. In Proceedings of VERIFY Workshop at FLoC, Copenhagen, Denmark, (2002)..

[22] OMG, UML 2.0 Superstructure Specification, (2005)

[23] Rumbaugh, J, Jacobson, G, Booch, G, The Unified Modeling Language Reference Manual Second Edition, Adison Wesley (2004)

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

[25] Thalheim, B. Fundamentals of Entity-Relationship Modeling. Annals Mathematics and Artificial Intelligence, 7, 197-256, (1993).

[26] Thalheim, B. Fundamentals of cardinality constraints. In Proceedings of the 11th International Conference on the Entity-Relationship Approach (ER-92), LNCS 645, 7-23, (1992).

[27] Heckel, R. and Voigt H. (2003). Towards consistency of web service architectures. In Proceedings of the 7th World Multiconference on Systemics, Cybernetics, and Informatics (SCI 2003).

[28] A. Ben Hadj Ali and F. Boufares and A. Abdellatif, On the global coherence of integrity constraints in UML class diagrams, Proceedings of the 24th IASTED international conference on Database and applications, (2006)

[29] Sunye, G., Pollet, D., Le Taraon, Y., and Jezkel J.-M. Refactoring UML models. In Proceedings of UML 2001, LNCS 2185, 134-148, (2001).

[30] Unhelkar, B.: Verification and Validation for Quality of UML 2.0 Models. Addison-Wesley,(2005)

[31] Szlenk, M. (2006) . Formal Semantics and Reasoning about UML Class Diagram, In IEEE Proceedings of the International Conference on Dependability of Computer Systems, 51-59.

[32] Kaneiwa, K. and Satoh, K. (2006). Consistency checking algorithms for restricted UML class diagrams. In Proceedings of the Fourth International Symposium on Foundations of Information and Knowledge Systems (FoIKS2006), LNCS 3861, Springer-Verlag, 219-239.

[33] Kim,S. and Carrington, D. Formalizing the UML Class Diagram Using Object-Z. Second International Conference on the Unified Modeling Language: Beyond the Standard, LNCS 1723, (1999).

[34] G.Genova, J.Llorens, P.Martinez. ,The meaning of multiplicity of n-ary associations in UML, Journal on Software and Systems Modeling, 2002

[35] Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., and Rosati, R. . Description Logic Framework for Information Integration. In Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR'98), 2-13, (1998).

[36] Richters, M. (2002). A Precise Approach to Validating UML Models and OCL Constraints. Phd thesis. Universitaet Bremen. Logos Verlag, Berlin, BISS Monographs, No. 14.

[37] Queralt, A. and Teniente, E. Reasoning on UML Class Diagrams with OCL Constraints. In Proceedings of 25th International Conference on Conceptual Modeling (ER2006), 497-512, (2006)..

[38] Evans, A. S. and Clark, T. (1997). Foundations of the Unified Modeling Language. In Proceedings of the 2nd Northern Formal Methods Workshop.

[39] Shroff, M. and France R. (1997). Towards a Formalization of UML Class Structures in Z. In Proceedings of the 21st International Computer Software and Applications Conference (COMPSAC'97), 646-651.

[40] Evans, A., France, R., Lano, K., and Rumpe, B. (1998). The UML as a formal modeling notation. In Proceedings of the Unified Modeling Language: the first international workshop

[41] Andri, P., Romanczuk, A., Royer, J.-C., and Vasconcelos, A. An algebraic view of UML class diagrams. In Proceedings of the 6th colloquium on Languages and Models with Objects, 261-276, (2000).

[42] Andre, P., Romanczuk-Requile, A., Royer, J.-C., and Vasconcelos, A. Checking the Consistency of UML Class Diagrams Using Larch Prover. In Proceedings of the third Rigorous Object-Oriented Methods Workshop, (2000).

[43] He, X. Formalizing UML class diagrams - a hierarchical predicate transition net approach. In Proceedings of the 24th Annual International Computer Software and Applications Conference, 217-222, (2000).

[44] Lano, K., Clark, D., and Androutsopoulos, K. UML to B: Formal Verification of Object-Oriented Models. In Integrated Formal Methods, 4th International Conference, IFM 2004, Canterbury, UK, April (2004).

[45] Jackson, D. and Rinard, M. (2004). Software analysis: a roadmap, In Proceedings of the Conference on the Future of Software Engineering, 133-145.

[46] Marcano, R. and Levy N. Using B formal specifications for analysis and verfication of UML/OCL models. Workshop on consistency problems in UML-based software development. 5th International Conference on the Unified Modeling Language. Dresden, Germany, October (2002)

[47] Garland, S. J. and Guttag, J. V. (1989). An overview of LP, the larch power. In Proceedings of the 3rd international Conference on Rewriting Techniques and Applications, 137-151

[48] Satoh, K., Kaneiwa, K., and Uno, T. (2006). Contradiction Finding and Minimal Recovery for UML Class Diagrams. In Proceedings of the 21st IEEE International Conference on Automated Software Engineering.

[49] Franconi, E. and Ng, G. (2000). The ICOM Tool for Intelligent Conceptual Modelling. In Proceeding of the Seventh International Workshop on Knowledge Representation meets Databases.

[50] Berardi, D. (2002). Using description logics to reason on UML class diagrams. In Proceeding of the KI'2002Workshop on Applications of Description Logics.

[51] Dullea, J. and Song, I. (1998). An Analysis of Structural Validity of Ternary Relationships in Entity-Relationship Modeling. In Proceedings of Seventh International Conference on Information and Knowledge Management (CIKM '98), 331-339.

[52] Dullea, J., Song, I., and Lamprou, I. (2003). An analysis of structural validity in entity-relationship modeling, Data and Knowledge Engineering, 47 (2), 167-205.

[53] Chen, P. P. The entity-relationship model-toward a unified view of data. ACM Transaction on Database Systems, 1(1), 9-36, (1976).

[54] Halpin, T.A. A Logical Analysis of Information Systems: Static Aspects of the Data-Oriented Perspective, Ph.D. thesis, Department of Computer Science, University of Queensland, Brisbane, Australia, (1989).

[55] Minsky, M. A Framework for Representing Knowledge. Technical Report. UMI Order Number: AIM-306., Massachusetts Institute of Technology, (1974).

[56] Williams H.P. (1993). Model Solving in Mathematical Programming. John Wiley and Sons

[57] Poseidon for UML, 2005. http://www.gentleware.com

[58] Wust, J. 2005. SDMetrics: The software design metrics tool for UML. www.sdmetrics.com.