Complexity boundaries for full satisfiability of restricted UML class diagrams

A dissertation submitted to

Faculty of Computer Science,

Free University of Bozen - Bolzano

in partial fulfillment of the degree of

Master of Science

Submitted by:

Yazmín Angélica Ibáñez García

Supervisors:

Prof. Alessandro Artale

Prof. Diego Calvanese

Freie Universität Bozen

Libera Università di Bolzano

Free University of Bozen · Bolzano

Fakultät für Informatik | Facoltà di Scienze e Tecnologie informatiche | Faculty of Computer Science

October 2009

Free University of Bozen - Bolzano

Author:	**Yazmín Angélica Ibáñez García**
Matriculation number:	**7533**
Title:	**Complexity boundaries for full satisfiability of restricted UML class diagrams**
Degree:	**Master of Science**
Date of submision:	**23.09.09**

Declaration
Hereby I certify that the thesis has been written by me. Any help that I have received in my research work has been acknowledged. Additionally, I certify that I have not used any auxiliary sources and literature except those I cited in the thesis.

Yazmín Angélica Ibáñez García

Abstract

During the last years, the Unified Modeling Language (UML) has emerged as the de-facto standard language in the object-oriented analysis and design world. UML Class diagrams are one of the most important components of UML. Class diagrams provide a static description of system components. These diagrams describe systems structure in terms of *classes*, *associations* (relations between classes), and constraints imposed on classes and their inter-relationships. It has long been argued that due to their lack of semantics, the effective use of graphical notations can be problematic when applied to the development of non-trivial systems. However, the semantics of UML class diagrams is by now well established, and one can exploit automated reasoning tools that are based on the languages underlying their formalization to detect relevant properties, such as class satisfiability and subsumption. A fundamental reasoning task is detecting full satisfiability of a diagram, i.e., whether *all* classes and associations of the diagram can be simultaneously populated without violating any constraints of the diagram. The importance of detecting full satisfiability of a UML class diagram is supported by the fact that unsatisfiable classes or associations indicate that the diagram contains unnecessary information, or that there is a modeling error, such as over constraining. One key element in the development of methods for implementing these reasoning tasks, is the study of the intrinsic complexity of such tasks. While the complexity of class satisfiability has been studied extensively, full satisfiability received less attention. In this thesis, we address the complexity of full satisfiability of UML class diagrams. Specifically, we establish tight upper and lower bounds for full satisfiability of UML class diagrams under different combinations of constraints. The results presented here confirm the intuition that full satisfiability has the same computational complexity as class satisfiability.

Acknowledgments

I would like to thank first of all to my supervisors Alessandro Artale and Diego Calvanese for their expert guidance, excellent comments and ideas that make this work possible.

I also would like to acknowledge the funding I received from the European Commission through the Erasmus Mundus Scholarship. This support made possible my stay in the EMCL program and give me the opportunity to meet wonderful people.

I would like to express my gratitude towards Carsten Lutz, for his comments and his help.

I wish to thank Prof. Steffen Hölldobler for his advice along my stay in the EMCL program.

Special thanks to my beloved Víctor, whose patience and support made possible this project. I appreciate all your efforts for making this work successful. Thanks for being there in the happy times, but more importantly for staying during the hard ones. Finally, all my love to my parents, my brothers and my sister that have been supporting me through the distance all these years.

Most importantly, thanks to God for giving me the strength for enduring the hardship to this life, and for putting in my way all the great people that make this work possible.

Angélica Ibáñez

Contents

Chapter 1

Introduction

The Unified Modeling Language (UML) is the de-facto industry standard for specifying, constructing, and documenting the artifacts of software systems. UML captures information about various aspects of a system by means of different visual modeling languages. *Class diagrams* (CDs) constitute one of the most important and best understood UML modeling languages. Class diagrams provide a static description of system components. These diagrams describe systems structure in terms of *classes*, *associations* (relations between classes), and constraints imposed on classes and their interrelationships. It has long been argued that the effective use of graphical notations can be problematic when applied to the development of non-trivial systems. A significant source of problems is their lack of *precise semantics*. However, the semantics of UML class diagrams is by now well established, and several works propose diverse formalizations of it (André et al., 2000a; Berardi et al., 2005; Gogolla and Richters, 2002; He, 2000; Kim and Carrington, 1999; Queralt and Teniente, 2006; Szlenk, 2006). Taking advantage of these formalizations, one can, in principle, reason on a given UML class diagram and prove properties about it. These properties serve as *quality* checks for the system represented by a diagram.

The quality of a system is largely determined early in the development cycle, i.e., during requirements specification and conceptual modeling. Moreover, errors introduced at these stages are usually much more expensive to correct than those of design or implementation. Thus, it is desirable to prevent, detect and correct errors as early as possible in the development process.

The principal justification for quality in conceptual models is that such models form the basis for designs and implementation. Well-understood, deterministic rules to transform a conceptual model into a design can be automated, in which case, a conceptual model of poor quality would result in an inadequate implementation.

The quality of class diagrams can be seen from two different points of

view. From an external point of view, quality refers to the correctness of the diagram regarding the user requirements. This can be validated, for instance, through checking whether the diagram specifies the relevant knowledge of the domain. From an internal point of view, quality can be determined by reasoning on the formalization of the CD, without taking requirements into account. In this sense, there are some typical reasoning tasks that can be performed on a CD like *satisfiability*, i.e., checking whether the information encoded in the diagram is not contradictory, *class satisfiability*, i.e., check whether there is an instantiation of the diagram in which a given class is populated. Even if a diagram is satisfiable, it may turn out that some class or association is empty in every legal instantiation. Satisfiability (liveliness) of classes or associations determines if a certain class or association can have at least one instance. More important, it is to determine whether all classes and associations can be instantiated simultaneously without violating the constraints imposed by the diagram. This is termed *full satisfiability*.

Full satisfiability is of importance as the presence of unsatisfiable (superfluous) classes or associations indicates that the diagram contains unnecessary information or that the constraints imposed on those classes/associations need to be relaxed.

The proposed formalizations for UML class diagrams allow to resort to automated inference procedures provided by the underlying formal language to verify the above mentioned properties. For example, the formalization in terms of *expressive description logics* provided in (Berardi et al., 2005) permits to use state-of-the-art description logics systems (Möller and Haarslev, 2003) to automatically verify such properties. Description logics are decidable logics specifically designed for representing knowledge in terms of classes and relationships between them. This makes these logics an ideal candidate for formalizing UML class diagrams. In fact, representing conceptual models by means of description logics has gained popularity over the past years, (c.f., Artale et al., 1996; Berardi et al., 2005; Borgida and Brachman, 2003; Calvanese et al., 1998b, 1999)

To avoid using a too powerful inference mechanism for reasoning on UML class diagrams, it is fundamental to investigate the complexity of reasoning on such diagrams, regardless of the formal tool adopted for describing them. The complexity of reasoning on UML class diagrams was addressed first in (Berardi et al., 2005), where it was shown that class satisfiability of UML class diagrams with completeness and disjointness constraints over class and associations generalizations is ExpTime-complete. The ExpTime lower bound for class satisfiability has been extended to diagrams with unconstrained generalizations over associations (and completeness constraints on class hierarchies only) in (Artale et al., 2007a). There, it is also shown that, by forbidding generalizations over associations, and completeness constraints the complexity of class satisfiability drops to NLogSpace.

However, no work has addressed explicitly the complexity of full satisfiability of UML class diagrams. In this thesis, we present both, upper and lower complexity bounds for full satisfiability of UML class diagrams that include different combinations of constrains.

1.1 Organization of this thesis

We now present a brief overview of the contents of the chapters of this thesis.

Chapter 2. This chapter presents a brief walkthrough of the development of Description Logics. Afterwards, it provides an introduction to description logics as a formal language for representing knowledge and for reasoning about it. The chapter gives a short overview of the ideas underlying description logics. Then it introduces the syntax and semantics of some basic constructs and how they are used to build knowledge bases. It also defines the typical inference problems in most description logics.

Chapter 3. In this chapter, we introduce UML class diagrams. The chapter includes a description of the main constructs used to build class diagrams, and a formalization of their semantics in terms of First Order Logic. The presentation is limited to the conceptual modeling aspects of class diagrams.

Chapter 4. This chapter addresses reasoning on UML class diagrams. The chapter includes topics such as the different reasoning tasks defined in the context of UML class diagrams, as well as the methods developed so far for implementing such tasks. Finally, it presents an overview of the complexity results established for reasoning on UML class diagrams.

Chapter 5. This chapter contains the proofs for the upper and lower complexity bounds of full satisfiability in UML class diagrams with different combinations of constraints. The results on this chapter provide a proof of the intuition that the complexity of full satisfiability coincides in all cases with that of class satisfiability. Indeed, the upper bounds are a direct consequence of the corresponding upper bounds for UML class diagrams derived from the description logic formalization, and the disjoint union model property of class diagrams. The lower bounds, on the other hand, are consequence of nontrivial adaptations of the proofs for the lower bounds for class satisfiability.

Chapter 6. This last chapter presents some conclusions of the work carried out in this thesis. It includes as well some future paths of research on the topic concerning this thesis.

Chapter 2

A brief review of Description logics

Description Logics (DL) (Baader et al., 2003) were developed in the field of Knowledge Representation as a formalism for providing a precise semantic characterization for frame systems (Minsky, 1974) and semantic networks (Quillian, 1968). Like semantic networks, frame systems and other network-based representation languages, description logics aim to represent sets of individuals and their relationships, but unlike the mentioned network-based formalisms, description logics are equipped with a formal logic-based semantics. This logic-based approach allows general purpose representation of the knowledge, and more importantly, it allows to reason about it, i.e., to infer implicit consequences of the explicitly represented knowledge.

One important step towards the development of description logics was the realization that network-based languages could be regarded as fragments of First Order Logic (FOL). As a result, different features of the representation language would lead to different fragments of FOL. The most important consequence of this fact is that the typical forms of reasoning used in structured based representations can be accomplished by specialized reasoning techniques, without necessarily requiring FOL theorem provers. Moreover, reasoning in different fragments of FOL leads to computational problems of different complexity.

Description logics have been developed, first as *terminological systems*, emphasizing that the representation language would establish the basic terminology in the domain of interest, and subsequently as *concept languages*, when the interest focused on the set of concept constructors admitted in the language. Finally, when the attention moved towards the properties of the underlying logical systems, the term *Description Logics* became popular.

DL has gone beyond its traditional scope in the Artificial Intelligence area to provide new alternatives and solutions in areas such as conceptual modeling, data integration and ontology-based data access (Borgida et al.,

2003; Calvanese et al., 1998b). Description logics are underlying the standard Web Ontology Language OWL, which is now in the process of being standardized by the W3C in its second edition, OWL 2.

In the conceptual modeling setting, the reasoning facilities of a DL system can be profitably exploited to support conceptual database design. The schema used in various semantic data models, can be translated into a description logic knowledge base in such a way that the verification of schema properties corresponds to traditional description logic reasoning tasks. The correspondence between semantic data models and description logics has been recently exploited to add advanced capabilities to CASE tools. A notable example is the I•COM tool (Franconi and Ng, 2000) for conceptual modeling, which combines a user-friendly graphical interface with the ability to automatically infer properties of a schema (e.g., inconsistency of a class, or implicit ISA relations) by invoking the FaCT description logic reasoner (Horrocks, 1998, 1999).

The rest of this chapter is organized as follows. We start by presenting the basic formalism of description logics in Section 2.1. We introduce the basic notions for describing concepts, and the means for representing knowledge in DL. The section concludes with an overview of the basic reasoning problems and how they are related to each other. Afterwards, Section 2.2 introduces the syntax and semantics of the basic DL language $\mathcal{ALC}$ and one of its syntactical variants. Finally, in Section 2.3, we introduce various DL languages specifically tailored for conceptual modeling and ontology representation.

2.1 Definition of the basic formalism

Description logics are well behaved FOL fragments equipped with decidable reasoning tasks. In general terms, a description logic is formed by three basic components:

- a *description language*,
- a *knowledge specification mechanism*,
- and a set of *automated reasoning tasks*.

2.1.1 DL languages

The main building blocks of all description logic languages are *atomic concepts* and *atomic roles*. Complex concepts can be built from them inductively using *concept constructors*. For example, intersection of concepts, which is denoted by $C \sqcap D$, is used to restrict the set of individuals under consideration to those that belong to both C and D. Note that, in the syntax of description logics concept expressions are variable-free. In fact, a

concept expression denotes the set of all individuals satisfying the properties specified by the expression.

The key characteristic features of DL reside in the constructs for establishing relationships between concepts. The basic ones are value restrictions. For example, a value restriction, written $\forall P.\, C$, requires that all the individuals that are in the relationship P with the concept being described belong to the concept C.

The set of constructors characterize the expressiveness of the description logic languages, and the complexity of the reasoning tasks. Various languages have been considered by the DL community, and numerous works investigate the relationship between expressive power and computational complexity of reasoning (see (Donini et al., 1996) for a survey).

The semantics of a DL language is given by a set-theoretic interpretation. A concept is interpreted as a set of individuals and roles are interpreted as sets of pair of individuals. The domain of an interpretation can be chosen arbitrarily, and it can be infinite. Formally, an *interpretation* $\mathcal{I}$ consists of a nonempty-set $\Delta^{\mathcal{I}}$ (the *domain* of the interpretation) and an interpretation function $\cdot^{\mathcal{I}}$, which a assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role name P a relation in $(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$.

More specifically, atomic concepts are interpreted as subsets of the domain, while the semantics of the complex concepts is specified by defining the set of individuals denoted by each construct.

For example, the concept $C \sqcap D$ is the set of individuals obtained by intersecting the sets of individuals denoted by C and D, respectively. Similarly, the interpretation of $\forall P.\, C$ is the set of individuals that are in the relationship P with individuals belonging to the set denoted by the concept C. Table 2.1, shows some constructors and their semantics.

Example 2.1. As a less abstract example, let us consider the domain of a university. Here, we are interested for example in representing the people involved in the university, their relations, as well as the courses in the university and their organization. Hence, we can have the following concept descriptions:

- $\mathsf{GraduateCourse} \sqcup \mathsf{UndergraduateCourse}$, for representing all the objects in the domain that are either graduate or undergraduate courses.
- $\mathsf{Course} \sqcap \forall\, \mathsf{enrolles}.\, \mathsf{Graduate}$, standing for all courses having only graduate students enrolled.
- $\mathsf{Academic} \sqcap \exists\, \mathsf{advises}.\, \mathsf{Graduate}$, for representing the academics that advise at least one graduate student. ◇

2.1.2 Knowledge Bases

A description logic *knowledge base* (KB) is composed by two elements: a *Terminological Box* (TBox), and an *Assertional Box* (ABox). The TBox contains intensional knowledge in the form of a terminology and it is built through declarations that describe general properties of concepts. The ABox contains extensional knowledge about the domain of interest, that is, assertions about individuals, usually called *membership assertions*.

One important element of a DL knowledge base is given by the operations used to build the terminology. Such operations are directly related to the forms and the meaning of the declarations allowed in the TBox.

The basic form of declaration in a TBox is a *concept definition*. A definition is an equivalence whose left-hand side is an atomic concept. Definitions are used to introduce *symbolic names* for complex descriptions. For example, by the axiom

$$A \equiv C \ ,$$

we associate to the description on the right-hand side the name A. Such a declaration is interpreted as a logical equivalence, which amounts to provide both sufficient and necessary conditions for classifying an individual as an A.

For example, we can use the following definition

$$\mathsf{Student} \equiv \mathsf{GraduateStudent} \sqcup \mathsf{UngraduateStudent} \ ,$$

to state that a student is either a graduate student, or an undergraduate student and nothing else.

There are some important common assumptions usually made about terminologies containing definitions of this form. In a TBox only one definition for each concept name is allowed. Furthermore, the definitions within the TBox are *acyclic*, in the sense that concepts are neither defined in terms of themselves nor in terms of other concepts that indirectly refer to them. This kind of restriction is common to many description logics and implies that every defined concept can be expanded in a unique way into a complex expression containing only atomic concepts by replacing every defined concept with the right-hand side of its definition. Although it has been shown that even simple expansions of definitions like the one mentioned is an important source of complexity (Nebel, 1990); in practice, definitions that inordinately increase the complexity of reasoning do not seem to occur.

It is also possible to drop the restrictions on the terminologies, in order to allow a more general setting for concept definitions, increasing the expressive power of the knowledge base. In particular, the admission of *cyclic* definitions has lead to different semantic interpretations of the declarations, known as *greatest/least fixed point*, and *descriptive* semantics. Although it has been argued that different semantics may be adopted depending on the

target application, the more commonly adopted one is descriptive semantics, which simply requires that all the declarations are satisfied in the interpretation. Moreover, by dropping the requirement that on the left-hand side of a definition there can only be an atomic concept name, one can consider so-called (general) *terminological axioms* of the form

$$C \sqsubseteq D \quad (R \sqsubseteq S) \qquad \text{(inclusion)}$$

$$C \equiv D \quad (R \equiv S) \qquad \text{(equivalence)}$$

where C and D are arbitrary concept expressions (and R,S are roles). Notice that a concept (role) *equivalence*, can be expressed by two *inclusions*.

The semantics of axioms is defined in a rather natural way. An interpretation $\mathcal{I}$ *satisfies* an inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I}$ *satisfies* a TBox $\mathcal{T}$ if and only if $\mathcal{I}$ satisfies each element of $\mathcal{T}$. If $\mathcal{I}$ satisfies an axiom (respectively, a TBox), then we say that it is a *model* of this axiom (respectively, TBox). Two axioms (or two TBoxes) are *equivalent* if they have the same models.

On the other hand, an ABox describes an specific state of affairs of an application domain in terms of concepts and roles. Some of the atomic concepts and roles in the ABox may be defined names of the TBox. In an ABox, one introduces individuals, by giving them names, and one asserts properties of these individuals. Using concepts C and roles R, one can make assertions of the following two kinds:

$$C(a) \qquad P(b, c)$$

where C and P denote a concept and a role respectively, and a, b, c denote individual names. The first kind of assertion, called *concept assertion* denotes that a belongs to (the interpretation of) C. The second kind, called *role assertion*, denotes that c is a filler of the role R for b.

For example, the assertion

$$\mathsf{Student}(\mathsf{VICTOR})$$

states that the individual VICTOR is a student. Similarly,

$$\mathsf{teaches}(\mathsf{STEPHAN}, \mathsf{TCS1})$$

specifies that the individual STEPHAN teaches the course TCS1.

We give a semantics to ABoxes by extending interpretations to individual names. In that case, an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ not only maps atomic concepts and roles to sets and relations, but in addition maps each individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. We assume that distinct individual names denote distinct objects. The interpretation $\mathcal{I}$ *satisfies* the concept assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and it *satisfies* the role assertion $P(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$.

An interpretation *satisfies* an ABox $\mathcal{A}$ if it *satisfies* each assertion in $\mathcal{A}$. In this case we say that $\mathcal{I}$ is a *model* of the ABox. Finally, $\mathcal{I}$ *satisfies* an assertion α (or an ABox $\mathcal{A}$) with respect to a TBox $\mathcal{T}$ if in addition to being a model of α (or of $\mathcal{A}$), it is also a model of $\mathcal{T}$.

2.1.3 Reasoning

The set-theoretic semantics of knowledge bases in description logics makes them equivalent to a set of axioms in FOL. Thus, like any other set of axioms, it contains implicit knowledge that can be made explicit through inferences. The automated reasoning tasks associated to a particular description logic are defined as logical inferences. These inferences can be defined for concepts, TBoxes, ABoxes or a combination of these elements.

Typical reasoning tasks for KBs are to determine whether a concept is *satisfiable* (i.e., non-contradictory), or whether one concept is more general than another one, that is, whether the first *subsumes* the second. Important reasoning tasks for an ABox are to find out whether its set of assertions is *consistent* (i.e., whether the ABox has a model); and whether the assertions in the ABox entail that a particular individual is an *instance* of a given concept description. Satisfiability checks of descriptions and consistency checks of sets of assertions are useful to determine whether a knowledge base is meaningful at all. With subsumption tests, one can organize the concepts of a terminology into a hierarchy according to their generality. A concept description can also be conceived as a query, describing a set of objects one is interested in. Thus, with instance tests, one can retrieve the individuals that satisfy the query. These reasoning tasks are formally defined as follows.

Definition 2.2 (Reasoning tasks). Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, C and D concept descriptions and a an individual name.

Satisfiability: C is *satisfiable* with respect to $\mathcal{K}$ if there exists a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}} \neq \emptyset$.

Subsumption: C is *subsumed by* a concept D with respect to $\mathcal{K}$ (denoted by $C \sqsubseteq_{\mathcal{K}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.

Equivalence: C and D are *equivalent* with respect to $\mathcal{K}$ (denoted by $C \equiv_{\mathcal{K}} D$) if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$.

KB consistency: $\mathcal{K}$ is *consistent* if there exists a model of both, $\mathcal{T}$ and $\mathcal{A}$.

instance a is an *instance* of C with respect to $\mathcal{K}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{K}$. $\diamond$

Constructor	Syntax	Semantics
negation	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
existential restriction	$\exists P.\, C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y.\, (x,y) \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction	$\forall P.\, C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y.\, (x,y) \in P^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$

Table 2.1: Syntax and semantics of $\mathcal{ALC}$ concept descriptions

2.2 $\mathcal{ALC}$ and $\mathcal{ALC}^-$

In this section, we present the $\mathcal{ALC}$ description logic language. $\mathcal{ALC}$ is a rather basic and well studied description logic. This logic includes Boolean constructors for concept descriptions, as well as value and existential restrictions.

We will use the letters A and B (possibly with subindex) to denote atomic concepts, the letter P to denote atomic roles, and the letters C and D for complex concept descriptions.

2.2.1 The description language $\mathcal{ALC}$

The basic elements of the $\mathcal{ALC}$ language are atomic concepts and (binary) roles. For expressing more complex concept descriptions, $\mathcal{ALC}$ contains Boolean constructors for concepts, as well as existential and universal restrictions for roles. More precisely, complex concepts in $\mathcal{ALC}$ are formed by the constructors shown in Table 2.1.

The union of concepts can be expressed naturally in terms of the intersection; $C_1 \sqcup C_2 := \neg(\neg C_1 \sqcap \neg C_2)$, and the value restriction in terms of the existential restriction; $\forall P.\, C := \neg\exists P.\, C$.

The formal semantics of $\mathcal{ALC}$ concepts, as usual for description logics, is specified in terms of an interpretation. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role name P a relation in $(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$. Table 2.1 shows the semantics of $\mathcal{ALC}$ concept descriptions.

An $\mathcal{ALC}$ TBox $\mathcal{T}$ is a finite set of inclusion axioms of the form

$$C_1 \sqsubseteq C_2 \ .$$

The semantics of axioms is defined in a natural way. An interpretation $\mathcal{I}$ *satisfies* an inclusion axiom $C_1 \sqsubseteq C_2$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.

An $\mathcal{ALC}$ ABox $\mathcal{A}$ is a finite set of assertions of the form $C(a)$ or $P(a, b)$, where C is a concept description, P a role name, and a, b are individual names.

A Knowledge Base (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ in $\mathcal{ALC}$ is conformed by a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$.

$\mathcal{ALC}$ reasoning tasks, in which we are interested, include concept satisfiability and concept subsumption w.r.t. a TBox, and KB satisfiability. Traditionally, the basic reasoning mechanism provided by DL systems checks the subsumption of concepts. This, in fact, is sufficient to implement also the other inferences, as the mentioned reasoning tasks are mutually reducible in polynomial time in $\mathcal{ALC}$. In fact, this is the case for all the description languages that include full negation and conjunction (i.e., that are Boolean complete).

One important issue in DL is the complexity of reasoning. This because description logics are conceived as a formal specification of knowledge to be used in knowledge base systems, This means, that one needs to derive implicit facts from the knowledge represented in an automated way. Hence, the implementation of derivation procedures should take into account the optimality of reasoning algorithms. In particular, reasoning in $\mathcal{ALC}$ is ExpTime-complete (Schmidt-Schauß and Smolka, 1991).

2.2.2 A restricted form of $\mathcal{ALC}$

A *primitive* inclusion assertion is an assertion of the form $A \sqsubseteq C$, i.e., an assertion containing an atomic concept in the left-hand side. A syntactically restricted form of $\mathcal{ALC}$ can be obtained by dropping intersections and allowing only for complex concepts built with at most one constructor of $\mathcal{ALC}$. This variant of $\mathcal{ALC}$ is called $\mathcal{ALC}^-$ (Berardi et al., 2005).

Complex concepts in $\mathcal{ALC}^-$ are constructed inductively from atomic concepts and atomic roles using the following rules:

$$C \ ::= \ A \ \mid \ \neg A \ \mid \ A_1 \sqcup A_2 \ \mid \ \exists P.\, A \ \mid \ \forall P.\, A$$

An $\mathcal{ALC}^-$ TBox is a finite set of primitive $\mathcal{ALC}^-$ inclusion assertions of the form $A \sqsubseteq C$ where C is an $\mathcal{ALC}^-$ concept. The main reasoning task that we will consider is atomic satisfiability, i.e., to check, given an atomic concept A and an $\mathcal{ALC}^-$ TBox $\mathcal{T}$, whether there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $A^{\mathcal{I}} \neq \emptyset$.

In spite of the restrictions, the complexity of reasoning in $\mathcal{ALC}^-$ remains ExpTime-complete, since concept satisfiability in $\mathcal{ALC}$ can be reduced to atomic concept satisfiability in $\mathcal{ALC}^-$ (Berardi et al., 2005).

2.3 Description logics for Conceptual Modeling

The term Conceptual Modeling (CM) refers to the task of specifying the structure of the data that is going to be manipulated within an application. CM has been an extensive part of the research in several areas, such as databases, information systems, software engineering, and knowledge representation.

Among the most successful approaches for CM, we find the object-oriented formalisms and semantic data models. Both of these formalisms, are based on the notion of class. These, so called, *class-based* languages express knowledge in terms of objects and classes.

Generally speaking, a *class* denotes a subset of the domain of discourse, and an *object*, an element of such domain. A class-based language allows to express relationships and constraints among classes.

The most commonly used industry-grade class-based conceptual modeling formalisms are ER and UML class diagrams. These formalisms come in various, mostly graphical, notational variants that share a common core, but also cater for specifics according to the intended applications.

Due to their nature, there is a tight correspondence between conceptual modeling formalisms, such as the ER model and UML class diagrams, and various description logics (Artale et al., 2007a; Berardi et al., 2005; Borgida and Brachman, 2003; Calvanese et al., 1998b, 1999). This correspondence allows to identify models described in ER or UML class diagrams, for example, with description logics KBs, and therefore to provide precise semantics for the models. Moreover, the reasoning capabilities of description logics can be advantageously used to achieve automatic checking of the quality of the conceptual models.

A variety of description logics has been used in the research area of conceptual modeling languages. The $\mathcal{DLR}$ family of description logics (Calvanese and De Giacomo, 2003) has been shown to be useful for unifying the semantics (Calvanese et al., 1999) of class-based languages, such as ER schema, and UML class diagrams. In order to use state-of-the-art DL systems to perform reasoning on UML CDs, the description logic $\mathcal{ALCQI}$ has been used to encode the diagrams in such a way that the soundness and completeness of the reasoning is preserved (Berardi et al., 2005). The less expressive *DL-Lite* description logics (Calvanese et al., 2005) have been used to provide reasoning capabilities to restricted ER schema and to establish complexity boundaries of reasoning on such schema (Artale et al., 2007a). In this section we present an overview of these DL languages, including syntax, semantics and complexity of the main reasoning tasks.

2.3.1 The $\mathcal{DLR}$ description logics

The $\mathcal{DLR}$ family of description logics (Calvanese et al., 1998a) represents a natural generalization of traditional description logics towards n-ary relations. These logics have been developed for providing a formal characterization of conceptual modeling languages such as ER schema and UML class diagrams. This formalization enables computational support for both integration of conceptual data models and automated satisfiability and consistency checking.

The basic elements of $\mathcal{DLR}$ are *atomic relations*, and *atomic concepts*. From these atomic elements, one can inductively construct arbitrary relations of arity between 2 and some $n_{max} \geq 2$, and arbitrary concepts. Complex concepts and relations, denoted by C and R respectively, are build according to the following syntax rules:

$$\begin{array}{rcccccccc} C & ::= & \top_1 & | & A & | & \neg C & | & C_1 \sqcap C_2 \quad | \quad (\leq k[i]R) \\ R & ::= & \top_n & | & P & | & (i/n : C) & | & \dot{\neg} R \quad | \quad R_1 \sqcap R_2 \end{array}$$

where i denotes a component of a relation (i.e., an integer between 1 and n_{max}), n denotes the *arity* of a relation (i.e., an integer between 2 and n_{max}), and k denotes a non-negative integer. Concepts and relations must be *well-typed*, which means that only relations of the same arity can be combined in expressions of the form $R_1 \sqcap R_2$, and $i \leq n$ whenever it denotes a component of a relation of arity n.

The semantics of $\mathcal{DLR}$ is specified through the usual notion of interpretation. $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each relation R of arity n a subset $R^{\mathcal{I}}$ of $(\Delta^{\mathcal{I}})^n$. More precisely, the following conditions are satisfied

$$\begin{aligned} \top_n^{\mathcal{I}} &\subseteq (\Delta^{\mathcal{I}})^n, \\ \top_1^{\mathcal{I}} &= \Delta^{\mathcal{I}}, \\ P^{\mathcal{I}} &\subseteq \top_n^{\mathcal{I}}, \\ A^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}}, \\ (i/n : C)^{\mathcal{I}} &= \{t \in \top_n^{\mathcal{I}} \mid t[i] \in C^{\mathcal{I}}\}, \\ (\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\ (\dot{\neg} R)\mathcal{I} &= \top_n^{\mathcal{I}} \setminus R^{\mathcal{I}}, \\ (C_1 \sqcap C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}, \\ (R_1 \sqcap R_2)^{\mathcal{I}} &= R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}, \\ (\leq k[i]R)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \sharp\{t \in R_1^{\mathcal{I}} \mid t[i] = a\} \leq k\}. \end{aligned}$$

Observe that $\top_1$ denotes the interpretation domain, while $\top_n$, for $n \geq 1$, does not denote the n-Cartesian product of the domain, but only a subset

of it that covers all relations of arity n. It follows from these observations, that the "$\dot{\neg}$" constructor on relations expresses *difference on relations*, rather than complement.

A TBox in $\mathcal{DLR}$ is a finite set of inclusion axioms of both, concepts and relations. The axioms on a $\mathcal{DLR}$ TBox have the form:

$$C_1 \sqsubseteq C_2 \qquad\qquad R_1 \sqsubseteq R_2,$$

where R_1 and R_2 have to be of the same arity.

An interpretation $\mathcal{I}$ *satisfies* $C_1 \sqsubseteq C_2$ ($R_1 \sqsubseteq R_2$) if and only if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ ($R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$).

There are four extensions of $\mathcal{DLR}$. For conceptual modeling, we will be mainly interested in $\mathcal{DLR}_{ifd}$. This logic captures most of the expressiveness of common conceptual modeling languages (c.f., Chapter 4).

Besides inclusion assertions, $\mathcal{DLR}_{ifd}$ knowledge bases allow for assertions expressing identification constraints and functional dependencies. An *identification assertion* on a concept C has the form:

$$(\textbf{id}\ \ C\,[i_1]R_1, \ldots, [i_h]R_h),$$

where each R_j (for $j \in \{1, \ldots h\}$) is a relation, and each i_j denotes one component of R_j. Intuitively, such assertions state that no two different instances of C agree on the participation to $R_1, \ldots, R_h$. More precisely, if a is an instance of C that is the i_jth component of a tuple t_j of R_j, for $j \in \{1, \ldots, h\}$, and b is an instance of C that is the i_jth component of a tuple s_j of R_j, for $j \in \{1, \ldots, h\}$, and for each j, t_j agrees with s_j in all components different from i_j, then a and b are the same individual.

A *functional dependency assertion* on a relation R has the form:

$$(\textbf{fd}\ \ R\, i_1, \ldots, i_h \leftarrow j),$$

where $h \geq 2$, and $i_1, \ldots, i_h$, j denote components of R. These kind of assertions express that two tuples R that agree on the components $i_1, \ldots, i_h$, must agree also in the j-th component .

Unary functional dependencies are ruled out in $\mathcal{DLR}_{ifd}$, since these lead to undecidability of reasoning (Calvanese et al., 2001a).

The semantics for identification and functional dependency assertions is defined as follows:

- An interpretation $\mathcal{I}$ *satisfies* the assertion (**id** $C[i_1]R_1, \ldots [i_h]R_h$) if for all $a, b \in C^{\mathcal{I}}$ and for all $t_1, s_1 \in R_1^{\mathcal{I}}, \ldots, t_h, s_h \in R_h^{\mathcal{I}}$, we have that:

$$\left.\begin{array}{l} a = t_1[i_1] = \cdots = t_h[i_h], \\ b = s_1[i_1] = \cdots = s_h[i_h], \\ t_j[i] = s_j[i], \text{ for } j \in \{1, \ldots, h\}, \text{ and for } i \neq i_j \end{array}\right\} \quad \text{implies} \quad a = b$$

- An interpretation $\mathcal{I}$ *satisfies* the assertion (**fd** $R\, i_1, \ldots, i_h \leftarrow j$) if for all $t, s \in R^{\mathcal{I}}$, we have that:

$$t[i_1] = s[i_1], \ldots, t[i_h] = s[i_h] \quad \text{implies} \quad t[j] = s[j].$$

Reasoning tasks in $\mathcal{DLR}_{ifd}$ include KB satisfiability, concept satisfiability w.r.t. a KB, concept subsumption and logical implication. Moreover, as this logic is Boolean complete, all these reasoning tasks can be polynomially reduced to each other. Hence, we can refer to all these tasks in general as *reasoning*. $\mathcal{DLR}_{ifd}$, as many description logics, has decidable reasoning tasks, i.e., it admits terminating reasoning procedures that are sound and complete with respect to the semantics. In particular, reasoning on $\mathcal{DLR}_{ifd}$ is ExpTime-complete (Calvanese et al., 2001a).

2.3.2 The description logic $\mathcal{ALCQI}$

$\mathcal{ALCQI}$ (Calvanese and De Giacomo, 2003) is a rich DL in which knowledge is represented in terms of concepts and binary relations (roles). $\mathcal{ALCQI}$ can be seen as a fragment of $\mathcal{DLR}$, where KBs are restricted to be a finite set of inclusion assertions on concepts only (no inclusion assertions on relations).

Let A and P denote, respectively, atomic concepts and atomic roles. $\mathcal{ALCQI}$ concepts and roles, respectively denoted by C and R, are built according to the following syntax rules:

$$\begin{array}{lcl} C & ::= & A \mid \neg C \mid C_1 \sqcap C_2 \mid (\leq k R.\, C) \\ R & ::= & P \mid P^- \end{array}$$

An $\mathcal{ALCQI}$ KB is constituted by a finite set of inclusion assertions of the form $C_1 \sqsubseteq C_2$, with C_1 and C_2 arbitrary $\mathcal{ALCQI}$ concepts.

The semantics of $\mathcal{ALCQI}$ constructs and KBs is analogous to that of $\mathcal{DLR}$. In particular, the semantics for *inverse roles* and *qualified number restrictions* is defined as follows:

$$\begin{array}{rcl} (P^-)^{\mathcal{I}} & = & \{(a, a') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (a', a) \in P^{\mathcal{I}}\} \\ (\leq k R.\, C)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \sharp\{(a, a') \in R^{\mathcal{I}} \wedge a' \in C^{\mathcal{I}}\} \leq k\} \end{array}$$

Reasoning tasks in $\mathcal{ALCQI}$ include KB satisfiability, concept satisfiability w.r.t. a KB, concept subsumption and logical implication. These reasoning tasks, as for $\mathcal{DLR}_{ifd}$, can be reduced to each other in polynomial time. Moreover, reasoning on $\mathcal{ALCQI}$ KB is also ExpTime-complete (Calvanese and De Giacomo, 2003).

2.3.3 The *DL-Lite* family

The logics of the *DL-Lite* family are specifically tailored for providing efficient access to large data repositories. *DL-Lite* DLs capture basic ontology

languages and keep low complexity of reasoning. Reasoning here means not only computing subsumption between concepts, and checking satisfiability of the whole knowledge base, but also answering complex queries over the set of instances maintained in secondary storage.

The basic elements of *DL-Lite* are *atomic concepts* and *atomic roles*. The minimum language of the *DL-Lite* family is denoted as *DL-Lite*$_{core}$. Basic and complex concepts, denoted by B and C respectively, are defined by the following rules:

$$\begin{array}{lcllll} R & ::= & P & | & P^- & \\ B & ::= & \bot & | & A & | \quad \exists R \\ C & ::= & B & | & \neg B & | \quad C_1 \sqcap C_2 \end{array}$$

DL-Lite$_{core}$ TBoxes are finite sets of assertions of the form $B \sqsubseteq C$.

Some of the extensions of the *DL-Lite*$_{core}$ language, that have been proposed in the context of conceptual modeling, are the logics *DL-Lite*$_{Bool}$, and its sublanguage *DL-Lite*$_{krom}$ (Artale et al., 2007b).

DL-Lite$_{Bool}$ *complex roles* and *concepts* are defined as follows:

$$\begin{array}{lcllll} R & ::= & P & | & P^- & \\ B & ::= & \bot & | & A & | \quad \geq q R \\ C & ::= & B & | & \neg C & | \quad C_1 \sqcap C_2 \end{array}$$

where $q \geq 1$.

A *DL-Lite*$_{Bool}$ knowledge base is a finite set of assertions of the form

$$C_1 \sqsubseteq C_2 \ ,$$

while for the *Krom* fragment, only axioms of the following forms are allowed:

$$\begin{array}{rcl} B_1 & \sqsubseteq & B_2 \\ B_1 & \sqsubseteq & \neg B_2 \\ \neg B_1 & \sqsubseteq & B_2 \end{array}$$

where B_1, B_2 are basic concepts (i.e., are of the form $\bot$, A, or $\geq qR$).

The main reasoning tasks for *DL-Lite* description logics are TBox satisfiability and concept satisfiability w.r.t. a TBox. As for the complexity of reasoning in the *DL-Lite* description logics, we have that TBox satisfiability is NP-complete for *DL-Lite*$_{Bool}$, and NLogSpace-complete for *DL-Lite*$_{krom}$ (Artale et al., 2007b).

Chapter 3

UML for class diagrams

During the last years, the Unified Modeling Language (UML)[1] has emerged as the de-facto standard language in the object-oriented analysis and design world. The initial versions of UML originated with three leading object-oriented methods (Booch (Booch, 1990), OMT (Rumbaugh et al., 1991), and CRC (Wirfs-Brock et al., 1990)), and incorporated a number of best practices from modeling language design, object-oriented programming, and architectural description languages. The UML went through a standardization process with the OMG (Object Management Group)[2] and is now an OMG standard (UML).

UML is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system. The modeling language is intended to unify past experience about modeling techniques and to incorporate current software best practices into a standard approach. UML includes semantic concepts, notation, and guidelines. It has static, dynamic, environmental, and organizational parts. It is intended to be supported by interactive visual modeling tools that have code generators and report writers. The objective of UML is to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes.

One of the primary goals of UML is to advance the state of the industry by enabling object visual modeling tool interoperability. However, to enable meaningful exchange of model information between tools, agreement on semantics and notation is required.

The static view is the foundation of UML. The elements of the static view of a model are the concepts that are meaningful in an application. The static view captures object structure. An object-oriented system unifies data structure and behavioral features into a single object structure. The

[1] http://www.omg.org/spec/UML/
[2] http://www.omg.org/

static view includes all the traditional data structure concerns, as well as the organization of the operations on the data. Both data and operations are quantized into classes.

The static view is notated using *class diagrams* (CDs). Class diagrams are probably the most important and best understood among all UML models. A CD shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships. UML CDs are used for generating code skeleton and database schemata, as well as a means for knowledge representation such as for specifying ontologies, and for defining meta-models of other programming, modeling, and specification languages. We introduce in Section 3.1 the notation of the main elements used in CD.

We will address UML CDs from the conceptual perspective. Specifically, we do not deal in our presentation with those features that are relevant for the implementation perspective, such as the qualifiers for operations and attributes: *public*, *protected*, and *private*.

The semantics of the various UML concepts and constructs is officially given by a description (in natural language) of the meaning of each construct. However, for the purposes of this thesis, a formal definition of the semantics results useful. There are various formalizations of the semantics for UML languages (André et al., 2000b; Berardi et al., 2005; Gogolla and Richters, 2002; He, 2000; Kim and Carrington, 1999; Shroff and France, 1997; Szlenk, 2006). Here, we present in Section 3.2 a formalization of the semantics of UML class diagrams in terms of fist order logic.

3.1 Class diagrams notation

A class diagram (CD) is a graphic representation of the overall structure of the domain of discourse that shows a collection of static model elements. The main constituents of this static view are *classes* and their relationships. Relationships among classes include *associations* and *generalizations*.

A class models a concept (set of elements) from the application domain. Classes are the center around which CDs are organized, while the other modeling elements are owned by or attached to classes. As for the relations among classes, associations in a CD describe semantic connections among individual objects of given classes, while generalizations relate general classes (superclasses) to more specialized (subclasses). Generalizations allow for an incremental description of the domain.

Classes

A *class* in a UML CD (see Figure 3.1) denotes a set of objects with common structure, behavior and relationships.

The notation for a class is a rectangle with three compartments:

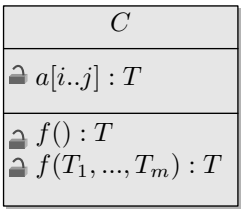


Figure 3.1: Class in UML

- the first compartment contains the *name* of the class,
- the second, *attributes* of the class, and
- the third, the *operations* of the class.

The compartments for attributes and operations can be omitted when full detailed description is not needed. Class names are unique within a diagram, while attributes (operations) are relative to a class, which means that two different classes may have two attributes (operations) with the same name.

Attributes are denoted by a name, possibly followed by the multiplicity and with the associated *type*. An attribute a of type T for a class C associates to each instance of C a set of instances of T. The optional multiplicity $[i..j]$ for a specifies that a associates to each instance of C at least i and at most j instances of T. When there is no upper bound on the multiplicity, the symbol $*$ is used for j. When the multiplicity is missing, [1..1] is assumed, that means that the attribute is *mandatory* and *single-valuated*.

An *operation* of a class represents a function from the objects of the class to which the operation is associated, and possibly additional parameters, to objects or values. An operation definition for a class C has the form

$$f(P_1, \ldots, P_m) : Q$$

where f is the name of the operation, $P_1, \ldots, P_m$ are the types of the *parameters*, and Q is the type of the result. As CDs represent the static view of the system, the actual definition of the function is not included in the diagram. Operations are represented by their *signature*, i.e., the name of the function and the number and the types of parameters, and the return type of the function.

Associations

Relations between instances of two or more classes are represented by *associations*. Names of associations (as names of classes) are unique in a UML CD. Associations carry information about relationships among the elements of the domain. Each connection of an association to a class is called an *association end*. Most information about an association is attached to one

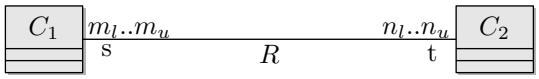


Figure 3.2: Binary association in UML

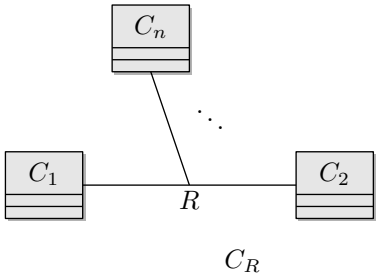


Figure 3.3: n-ary association in UML

of its ends. Association ends can have attached names (*rolenames*) and *multiplicities*. All the rolenames in an association must be different. Within a self-association (an association involving the same class more than once), rolenames are necessary to disambiguate the ends attached to the same class. Otherwise, rolenames are optional, because the class names can be used to disambiguate the ends. Alternatively, the association name can be omitted, and then the rolenames on its ends provide an alternative way of distinguishing multiple associations among the same classes.

The most common kind of association is a *binary association*. The notation for a binary association is a line or path connecting the participating classes. A binary association R between two classes C_1 and C_2 is graphically rendered as a solid line (see Figure 3.2). The association name is placed along the line with the rolename and multiplicity at each end. The *multiplicity* constraint $n_l..n_u$ on one of the binary association ends specifies that each instance of the class C_1 can participate at least n_l times and at most n_u times to relation R; analogously for $m_l..m_u$ and C_2. When the multiplicity is omitted, it is intended to be $0..*$. Multiplicity is most useful for binary associations because its definition for n-ary associations is complicated. An n-ary association $C_1, C_2, \ldots, C_n$, is depicted in Figure 3.3.

An association can also have attributes of its own, in which case it is both an association and a class i.e., and *association class*. A binary association with association class C_R, is rendered in Figure 3.4. An association class can also be added to an n-ary association, as in Figure 3.5.

Aggregations are a particular kind of binary associations. An aggregation denotes a part-whole relationship, i.e., a relationship that specifies that each instance of a class contains a set of instances of another class. Aggregations have no associated class. An association is graphically denoted by a hollow-diamond adornment on the end of the path attached to the *aggregate class*.

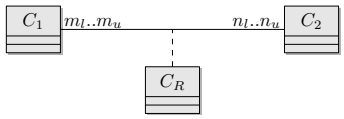


Figure 3.4: Binary association with association class in UML

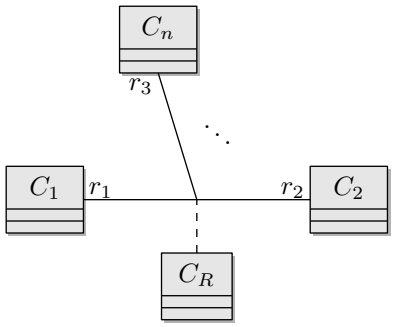


Figure 3.5: n-ary association with association class

An aggregation is graphically rendered as shown in Figure 3.6.

Generalizations

The *generalization* relationship is a taxonomic relationship between a more general class and a more specific class that builds on it and extends it. The more specific class is completely consistent with the more general one (it has all its properties, members, and relationships) and may contain additional information. The more general class is called the *parent* class (*superclass*); an element in the transitive closure is an *ancestor*. The more specific class is called the *child* class (*subclass*); an element in the transitive closure is a *descendant*. Since a generalization specifies that each instance of the child class is also an instance of the parent class, the instances of the child class *inherit* the properties of the parent class. The inheritable properties include attributes, operations, constraints, and participation in associations.

A generalization is graphically denoted as an arrow from the child to the parent class, with a large hollow triangle on the end connected to the parent. Several generalizations can be grouped together to form a class *hierarchy*, as shown in Figure 3.7.

A generalization relation can be also expressed between two associations.

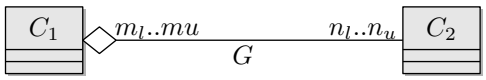


Figure 3.6: Aggregation in UML

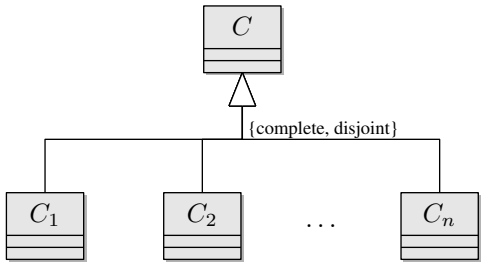


Figure 3.7: A class hierarchy in UML

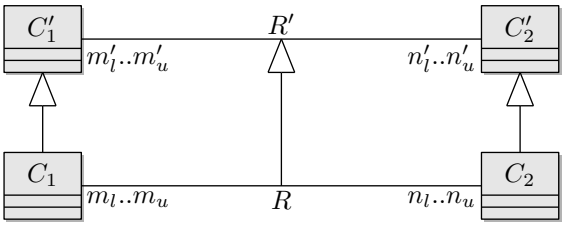


Figure 3.8: Association generalization in UML

The notation for generalizations among associations is the same as the one for a generalization between two classes. Figure 3.8 shows a generalization of binary associations.

Constraints

UML supplies a set of concepts and relationships for modeling systems as graphs of modeling elements. Some things, however, are better expressed using a textual language. A constraint is a Boolean expression represented as a string to be interpreted in a designated language. Natural language, set theoretic notation, constraint languages, or various programming languages may be used to express constraints. The UML includes the definition of a constraint language, called OCL, that is convenient for expressing UML constraints and is expected to be widely supported. See (Warmer and Kleppe, 1999) for more information on OCL.

Constraints can be used to state various nonlocal relationships, such as restrictions on associations.

Some standard constraints are predefined as UML standard elements, including associations in an exclusive-or relationship and various constraints on the relationships of subclasses in generalizations.

A constraint is shown as a text expression in braces. It may be written in a formal language or natural language. A constraint may be applied to a set of generalization relationships and their children that share a common parent. The following properties can be specified.

disjoint	No element may have two children in the set as ancestors (in a multiple inheritance situation). No instance may be a direct or indirect instance of two of the children (in a multiple classification situation).
overlapping	An element may have two or more children in the set as ancestors. An instance may be an instance of two or more subclasses.
complete	All possible children have been enumerated in the set and no more may be added.
incomplete	All possible children have not been enumerated yet in the set.

Another kind of constraints are multiplicities. Multiplicity specifications may be given for association ends. The multiplicity attached to an association end declares how many objects may fill the position defined by the association end. For a binary association, the multiplicity on the target end constrains how many objects of the target class may be associated with a given single object from the other (source) end. Multiplicity is typically given as a range of integers of the form minimum..maximum. Common multiplicities include [1..1](exactly one); [0..∗](zero or more, without limit); and [1..∗] (one or more, without limit).

Multiplicity constraints can also be applied to a class. In this case a multiplicity constraint declares how many instances of the class may exist. The usual default is unlimited, but a finite multiplicity is useful in some cases, particularly to declare a singleton class, that is, a class that may have only one instance. Multiplicity of a class is shown by placing a multiplicity string in the upper right corner of the rectangle symbol. The string may be omitted if the multiplicity is unlimited.

A more general form of multiplicity is the so called refinement of multiplicity constraints for sub-classes participating in associations. With such a construct one is able to change (and thus refine) the multiplicity constraints for sub-classes.

We conclude this section with an example of a UML class diagram that includes most of the constructs mentioned.

Example 3.1. To give a precise idea of how class diagrams can be used to model, let us take the domain of a university system. In Figure 3.9, we present a class diagram containing most of the constructs previously described. This diagram partially specifies a university system. It captures the people hierarchy within the university and their relationship to the university courses. The Academic class represents the set of academic people in the university, and the association between FacultyMember and Course denotes

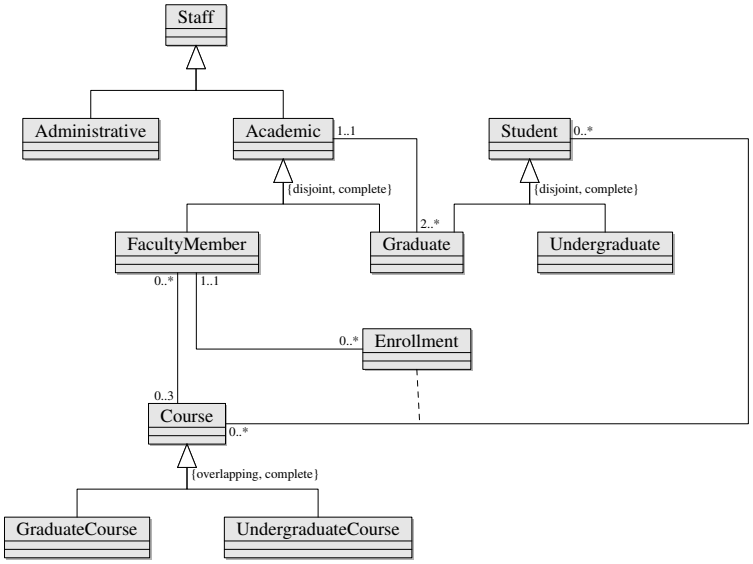


Figure 3.9: partial class diagram of a university system

a set of pairs of faculty members and courses in which the Faculty Member object plays the role of a teacher. The diagram expresses the constraint that a FacultyMember can teach up to three Courses (as indicated by the 0..3 multiplicity constraint). The class association Enrollment is used to represent the set of Course-Student pairs, so no two enrollments are identified by the same instance of Enrollment. Generalizations are used to specify that FacultyMember and Graduate are both subsets of Academic. The constraint {disjoint, complete} expresses that there is no other instances of Academic than those of FacultyMember and Graduate, and that no faculty member is a graduate and vice versa. Another example of constraints over a hierarchy is the use of the constraints {overlapping, complete} on the hierarchy of Course, to indicate that a course may be simultaneously a graduate and an undergraduate course, and that every course is either a graduate or an undergraduate course. ◇

3.2 Class diagrams semantics

Officially, the semantics of the constructs used in UML CDs are defined using natural language. However, as we will see in Chapter 4, it results very useful to have a formal definition of the semantics. We will present one of the most common formalizations of UML CDs (Berardi et al., 2005). We restrict the presentation to the constructs that are relevant for the scope of this thesis. More precisely, we present the formalization for classes, binary associations (with and without related class), and disjointness and completeness

constraints over generalizations.

We describe the semantics of each construct of UML CDs in terms of First Order Logic (FOL). This means that we will identify a *legal instantiation* of a UML diagram $\mathcal{D}$, with the model $\mathcal{I}$ of the corresponding set of FOL assertions $\Sigma(\mathcal{D})$.

Classes

A class C formally corresponds to a unary predicate C. Each class attribute a of type T formally corresponds to a binary predicate a for which the following FOL assertion holds:

$$\forall x, y.\, (C(x) \wedge a(x, y)) \rightarrow T(y)$$

The multiplicity $[i..j]$ associated to the attribute a can be expressed by

$$\forall x.\, C(x) \rightarrow (\exists_{\geq i}\, y.\, a(x, y) \wedge \exists_{\leq j}\, y.\, a(x, y))$$

we use counting quantifiers to abbreviate the FOL formula encoding the cardinality constraints on the set of attribute values. These constraints indicate that there is at least i, and at most j distinct values for y such that $a(x, y)$ holds.

Associations

An association R between the classes C_1 and C_2 (without related class) corresponds formally to a binary predicate R that satisfies the following FOL assertion:

$$\forall x_1, x_2.\, R(x_1, x_2) \rightarrow C_1(x_1) \wedge C_2(x_2)\ .$$

Multiplicity constraints are formalized by the following assertions:

$$\begin{aligned} \forall x.\, (C_1(x) &\rightarrow \exists_{\geq n_l} y.\, R(x, y) \wedge \exists_{\leq n_u} y.\, R(x, y))\,, \\ \forall y.\, (C_2(y) &\rightarrow \exists_{\geq m_l} x.\, R(x, y) \wedge \exists_{\leq m_u} x.\, R(x, y))\,. \end{aligned}$$

Refinement involving a binary association, R, between classes C_1, C_2 and a subclass of C_1, say C_1', can be formalized with the following FOL assertions:

$$\begin{aligned} \forall x.\, (C_1'(x) &\rightarrow C_1(x))\,, \\ \forall x.\, (C_1'(x) &\rightarrow \exists_{\geq n_l'} y.\, R(x, y) \wedge \exists_{\leq n_u'} y.\, R(x, y))\ . \end{aligned}$$

A binary association with a related association class C_R is formalized in FOL by reifying the association into a unary predicate C_R with two binary

predicates P_1, P_2, one for each component of the association. The semantics of the association is formalized by the above FOL assertions. For $i = 1, 2$:

$$\forall x.(C_R(x) \rightarrow \exists y.\, P_i(x, y))\,,$$

$$\forall x, y.(C_R(x) \wedge P_i(x, y) \rightarrow C_i(y))\,,$$

$$\forall x, y, y'.(C_R(x) \wedge P_i(x, y) \wedge P_i(x, y') \rightarrow y = y')\,,$$

$$\forall y_1, y_2, x, x'.(C_R(x) \wedge C_R(x') \wedge \textstyle\bigwedge_{i\in\{1,2\}} P_i(x, y_i) \wedge P_i(x', y_i) \rightarrow x = x')\,.$$

For associations with a related class, the multiplicity constraints are formalized by the following FOL assertions:

$$\forall y_1.(C_1(y_1) \rightarrow \exists_{\geq n_l} x.\,(C_R(x) \wedge P_1(x, y_1)) \wedge \exists_{\leq n_u} y.\,(C_R(x) \wedge P_1(x, y_1)))\,,$$

$$\forall y_2.(C_2(y_2) \rightarrow \exists_{\geq m_l} x.\,(C_R(x) \wedge P_2(x, y_2)) \wedge \exists_{\leq m_u} x.\,(C_R(x) \wedge P_2(x, y_2)))\,.$$

Class generalization

Them semantics of a generalization between the classes C_1 and C_2 is formally captured by the FOL assertion:

$$\forall x.\, C_1(x) \rightarrow C_2(x)$$

The semantics of constraints over class generalizations are also captured by FOL. *Disjointness* among the classes $C_1, \ldots, C_n$ is expressed by

$$\forall x.\, C_i(x) \rightarrow \bigwedge_{j=i+1}^{n} \neg C_j(x) \qquad \text{for } i = 1, \ldots, n-1\,.$$

Completeness constraints, on the other hand are formalized by the assertion

$$\forall x.\, C(x) \rightarrow \bigvee_{i=1}^{n} C_i(x)\,.$$

Example 3.2. Figure 3.10 shows the FOL formalization of the class diagram shown in Figure 3.9. ◇

Generalizations

$$\forall x.\, \mathsf{Administrative}(x) \rightarrow \mathsf{Staff}(x)$$
$$\forall x.\, \mathsf{Academic}(x) \rightarrow \mathsf{Staff}(x)$$

$$\forall x.\, \mathsf{Graduate}(x) \rightarrow \mathsf{Student}(x)$$
$$\forall x.\, \mathsf{Undergraduate}(x) \rightarrow \mathsf{Student}(x)$$
$$\forall x.\, \mathsf{Undergraduate}(x) \rightarrow \neg\mathsf{Graduate}(x)$$
$$\forall x.\, \mathsf{Student}(x) \rightarrow \mathsf{Graduate}(x) \lor \mathsf{Undergraduate}(x)$$

$$\forall x.\, \mathsf{Graduate}(x) \rightarrow \mathsf{Academic}(x)$$
$$\forall x.\, \mathsf{FacultyMember}(x) \rightarrow \mathsf{Academic}(x)$$
$$\forall x.\, \mathsf{Graduate}(x) \rightarrow \neg\mathsf{FacultyMember}(x)$$
$$\forall x.\, \mathsf{Academic}(x) \rightarrow \mathsf{FacultyMember}(x) \lor \mathsf{Graduate}(x)$$

$$\forall x.\, \mathsf{GraduateCourse}(x) \rightarrow \mathsf{Course}(x)$$
$$\forall x.\, \mathsf{UndergraduateCourse}(x) \rightarrow \mathsf{Course}(x)$$
$$\forall x.\, \mathsf{UndergraduateCourse}(x) \rightarrow \neg\mathsf{GraduateCourse}(x)$$

Associations

$$\forall x, y.\, A(x, y) \rightarrow \mathsf{Academic}(x) \land \mathsf{Graduate}(y)$$

$$\forall x, y.\, T(x, y) \rightarrow \mathsf{FacultyMember}(x) \land \mathsf{Course}(y)$$

$$\forall x, y.\, O(x, y) \rightarrow \mathsf{Academic}(x) \land \mathsf{Enrollment}(y)$$

$$\forall x, y.\, \mathsf{Enrollment}(x) \land r_1(x, y) \rightarrow \mathsf{Course}(y)$$
$$\forall x, y.\, \mathsf{Enrollment}(x) \land r_2(x, y) \rightarrow \mathsf{Student}(y)$$
$$\forall x.\, \mathsf{Enrollment}(x) \rightarrow \exists y.\, r_1(x, y)$$
$$\forall x.\, \mathsf{Enrollment}(x) \rightarrow \exists y.\, r_2(x, y)$$
$$\forall x, y, y'.\, \mathsf{Enrollment}(x) \land r_1(x, y) \land r_1(x, y') \rightarrow y = y'$$
$$\forall x, y, y'.\, \mathsf{Enrollment}(x) \land r_2(x, y) \land r_2(x, y') \rightarrow y = y'$$
$$\forall x, x', y_1, y_2.\, \mathsf{Enrollment}(x) \land \mathsf{Enrollment}(x') \land r_1(x, y_1) \land r_1(x', y_1) \land r_2(x, y_2) \land r_2(x', y_2) \rightarrow x = x'$$

Multiplicities

$$\forall x.\, \mathsf{Academic}(x) \rightarrow \exists_{\geq 2}\, y.\, A(x, y)$$
$$\forall y.\, \mathsf{Graduate}(y) \rightarrow (\exists_{\geq 1}\, x.\, A(x, y) \land \exists_{\leq 1}\, x.\, A(x, y))$$

$$\forall x.\, \mathsf{FacultyMember}(x) \rightarrow \exists_{\leq 3}\, y.\, T(x, y)$$

$$\forall y.\, \mathsf{Enrollment}(y) \rightarrow (\exists_{\geq 1}\, x.\, O(x, y) \land \exists_{\leq 1}\, x.\, O(x, y))$$

Figure 3.10: FOL formalization of the class diagram shown in Figure 3.9

Chapter 4

Reasoning on UML class diagrams

The central role that UML CDs play in the design and specification of software, databases and ontologies makes evident the need of powerful tools that help the designer to detect redundancies and contradictions in a diagram. A promising approach for providing this kind of support for software designers and ontology engineers, is to integrate reasoning services within Computer Aided Software Engineering (CASE) tools. However, in order to reason about UML CDs, formal semantics of class diagrams needs to be defined. UML officially lacks of formal semantics, and hence, there is no precise way for making inferences from a given CD. One way to define semantics is to directly assign a denotation to every language construct. This is termed *declarative direct semantics*. An alternative way is to use a different formally defined language that already has a formal semantics as a mediator, this *indirect semantics* relies on a translation of the constructs of the language to expressions in the intermediate language.

The semantics of UML CDs has been defined using both approaches. The direct semantics of CDs assigns set extensions to classes and associations. Direct semantics for UML CDs has been formally defined by several studies (Richters, 2002; Szlenk, 2006). There are various indirect formalizations of the semantics for CDs in the literature. The use of First Order predicate logic as formalization for UML class diagrams is probably the most popular formalization. FOL has been used to formalize UML CDs with OCL constraints (Queralt and Teniente, 2006) and also without them (Berardi et al., 2005). Other indirect formalizations include the use of description logics as intermediate language. The description logic $\mathcal{DLR}_{ifd}$ has been used to formalize UML CDs without OCL constraints (Berardi et al., 2005). Some other intermediate languages used to formalized UML CDs are Z and Object-Z (Kim and Carrington, 1999; Shroff and France, 1997). Algebraic specifications in the form of Data types (André et al., 2000a,b), and Hi-

erarchical Predicate Transitions Nets (He, 2000) have also been studied as formalizations for UML CDs.

Once the semantic of UML CDs is established, one can *validate* a given diagram. This means, check whether the diagram fulfills the requirements imposed by the application. Since a UML CD is a model of a real domain, there can be different levels of abstraction according to the intended application of the model. In any case, this abstraction has to be correct, in the sense that it truthfully represents the reality. By its own nature, a model cannot be complete, as some parts of the reality may be considered less important during the modeling process. Therefore, it is crucial to identify the relevant properties to be checked so as to asses the quality of the model according to objective quality criteria.

These relevant properties, that serve as quality checks in a CD, can be regarded as reasoning tasks in the formalization of such a diagram. This because typically these formalizations have means for reasoning about the represented knowledge, as it is the case for FOL and description logics.

In the next Section, we will present the main reasoning tasks studied so far in the literature. Afterwards, in Section 4.2, we describe some of the methods that have been used to implement those reasoning tasks in practice. To complete the presentation, we discuss in Section 4.3 the complexity results established for reasoning on UML class diagrams.

4.1 Reasoning tasks on UML class diagrams

In order to analyze the validity of a constructed diagram, it is essential to identify the semantic properties of the diagram that ensure not only that the diagram is correct, but also that it faithfully represents the underlying reality.In this section, we define various properties of interest (see e.g. Borgida et al., 2003; Calvanese et al., 1998b) in terms of the FOL formalization presented in Section 3.2.

Given that the semantics of a UML class diagram $\mathcal{D}$ can be captured by a set of FOL assertions $\Sigma(\mathcal{D})$, it is possible to identify a *legal instantiation* of the diagram with the model of the corresponding set of assertions $\Sigma(\mathcal{D})$. Moreover, it is possible to identify the *instantiation* of a class or association with the corresponding predicate extension, and then say that a class is *populated* if the corresponding extension is non-empty. Taking the FOL formalization as a reference, one can define a number of relevant properties to be checked in a UML class diagram:

Definition 4.1 (Reasoning tasks). Let $\mathcal{D}$ be a UML CD, and $\Sigma(\mathcal{D})$ its FOL formalization.

Diagram satisfiability. $\mathcal{D}$ is *satisfiable*, if $\Sigma(\mathcal{D})$ admits a model. Intuitively, this means that there exists a legal instantiation of $\mathcal{D}$.

Class satisfiability. A class C within $\mathcal{D}$ is *satisfiable* if $\Sigma(\mathcal{D})$ admits a model $\mathcal{I}$, such that $C^{\mathcal{I}} \neq \emptyset$; i.e., there is a legal instantiation of $\mathcal{D}$ in which C can be populated.

Class subsumption. A class C_1 *subsumes* a class C_2, if in every model $\mathcal{I}$ of $\Sigma(\mathcal{D})$, $C_2^{\mathcal{I}} \subseteq C_1^{\mathcal{I}}$; i.e., the constraints imposed by $\mathcal{D}$ imply that C_1 is a generalization of C_2.

Class equivalence. Two classes C_1 and C_2 are *equivalent* in $\mathcal{D}$, if in every model $\mathcal{I}$ of $\Sigma(\mathcal{D})$, $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$; i.e., C_1 and C_2 denote the same set of instances whenever the requirements imposed by $\mathcal{D}$ are satisfied.

Implicit consequences More generally, a property is an (implicit) *consequence* of a CD, if it is a logical consequence of $\Sigma(\mathcal{D})$, i.e., the property holds in every model of $\Sigma(\mathcal{D})$. ◇

Each one of these properties covers a different need in the quality check of a model. Subsumption checking allows one to deduce that properties for C_1 hold also for C_2. This suggests either an omission of an explicit generalization, or a design error. Alternatively, if all instances of the more specific class are not supposed to be instances of the more general class, then there is an error in the diagram, since it is forcing an undesired conclusion. Class subsumption is also the basis for a classification of all the classes in a diagram. Such a classification, as in any object-oriented approach, can be exploited in several ways within the modeling process.

Example 4.2. The class diagram shown in Figure 4.1 models the relationships of the people in a university. The people are divided in two disjoint classes: Academics and Students. There are two kinds of students, graduate students and undergraduate students. One special kind of graduate students is the class of PhD students. On the other side, we have that some people have special relations with some structures within the university. Each academic is affiliated to exactly one department within the university, and each PhD student is also affiliated to one department by a PhD position, which is a special kind of affiliation.

We have that from the (explicit) subsumption between the association classes Affiliation and PhD Position, it follows that PhD Student is (implicitly) subsumed by the class Academic. This subsumption may lead to a contradiction in some instantiations of the diagram, namely in those where the classes PhD Position and PhD Student are nonempty, since PhD Student is subsumed by Student and no instance of this class is an instance of Academic, as imposed by the {disjoint} constraint in the Person hierarchy. This, clearly does not reflect truthfully the underlying reality, and hence, is a modeling error. The modeling error in the diagram could then be avoided by either eliminating the generalization between Affiliation and Position, or by relaxing the disjointness between Academic and Student. ◇

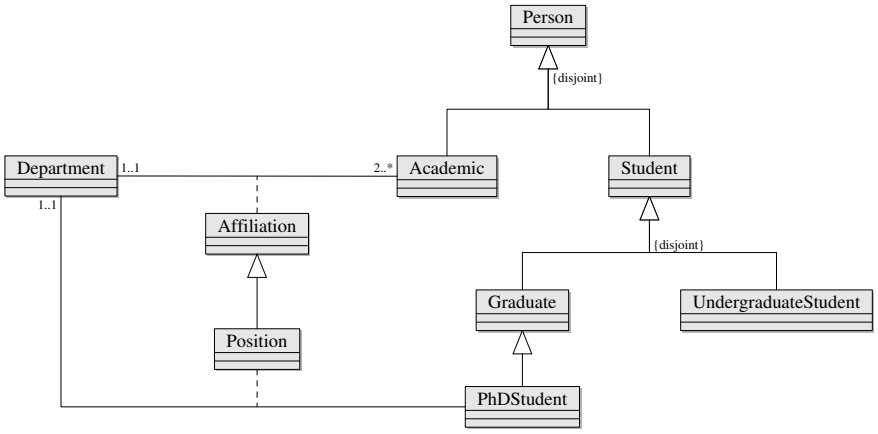


Figure 4.1: Subsumption in class diagrams

Determining equivalence of two classes allows for their merging, thus reducing the complexity of the diagram. Moreover, knowing about class equivalences avoids misunderstanding among different users.

Inferring implicit consequences is useful (according to the modeling needs) to reduce the complexity of the diagram by removing those parts that implicitly follow from other ones, or alternatively to make properties explicit, thus enhancing the readability of the diagram.

When a diagram is not satisfiable, the definitions altogether are contradictory. However, diagram satisfiability of a UML diagram only refers to the existence of a model of the diagram (or less abstract, some population of the diagram) as a whole. Such a model does not need to satisfy any classes or associations per se. The only condition is that all constraints are satisfied by the (possibly empty) populations.

On the other hand, class satisfiability ensures that the diagram has a model in which at least one class is populated. This mean that class satisfiability implies diagram satisfiability. The inconsistency of a class may be due to a design error or due to over constraining. In any case, if a class is unsatisfiable, the quality of the diagram is affected, since the class stands for the empty class, which at the very least means that the class is inappropriately named. To increase the quality of the diagram, the designer may remove the inconsistency by relaxing some constraints, or by deleting the class, thus removing redundancy.

Example 4.3. In the diagram shown in Figure 4.2 the class Graduate is unsatisfiable. Every possible instance of Graduate is also an instance of Academic and Student simultaneously. This, clearly, violates the disjointness constraint in the hierarchy. ◇

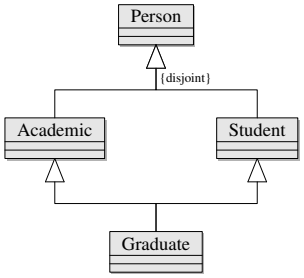


Figure 4.2: Unsatisfiability of UML class diagrams

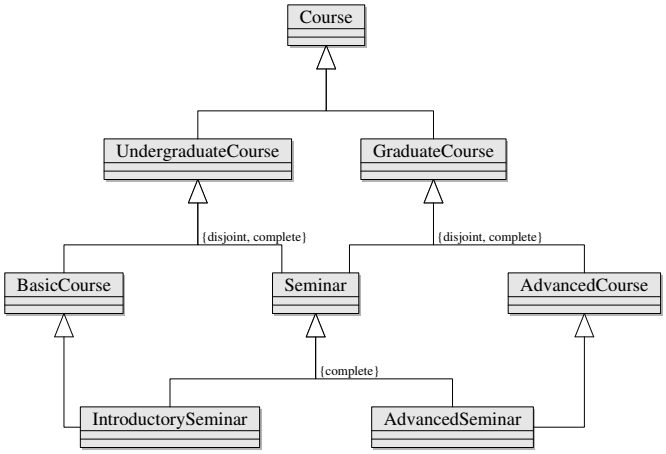


Figure 4.3: class diagram with unsatisfiable classes

4.1.1 Full satisfiability

Although diagram satisfiability and class satisfiability ensure important properties of a CD, a more fundamental property of CDs needs to be considered: to check whether there is a model of a UML diagram that satisfies *all* classes and *all* associations in a diagram simultaneously. This notion of satisfiability, referred here as *full satisfiability*, and introduced in (Kaneiwa and Satoh, 2006), is clearly a stronger notion than class satisfiability and diagram satisfiability. A model of a diagram that satisfies all classes is, by definition, also a model of that diagram and a witness that every class is satisfiable. Note that full satisfiability of a UML diagram (in which every class participates in at least one association) can be ensured by checking whether there is a model of the diagram in which all associations can be populated. This means that satisfiability of "all associations" is thus stronger than satisfiability of "all classes". Formally, we have the following definition.

Definition 4.4 (UML Full Satisfiability). A UML CD $\mathcal{D}$ is *fully satisfiable* if there is an FOL interpretation $\mathcal{I}$ that satisfies all the constraints

expressed in $\mathcal{D}$ and such that $C^{\mathcal{I}} \neq \emptyset$ for every class C in $\mathcal{D}$, and $R^{\mathcal{I}} \neq \emptyset$ for every association R in $\mathcal{D}$. We say that $\mathcal{I}$ is a *full model* of $\mathcal{D}$. $\diamond$

Full satisfiability of a CD is a fundamental property that a diagram should fulfill. The presence of some unsatisfiable class or association in a diagram can be considered as a modeling error. This because unnecessary information it is represented, since the unsatisfiable class/association stands for an empty set of instances or a relation in which no element of the domain participates. Another possible explanation for class/association unsatisfiability could be over constraining. Indeed, representing unnecessary information is harmless in certain applications, but in practice, it makes no sense to invert time in the implementation of a system in which some classes are never instantiated, or a database where some tables would always be empty.

Consider as an example again the diagram in Figure 4.1, although the diagram is satisfiable, there is no legal instantiation of the diagram in which every class is populated. The class PhD Student cannot be instantiated: the existence of a PhD student, implies the existence of a PhD Position, but then this implies that such a PhD student is an Academic and a Student at the same time which is not possible by the disjointness between Student and Academic.

As another example, consider the diagram shown in Figure 4.3, the constraints imposed by the diagram force the classes Seminar and its subclasses to be empty in every possible instantiation.

Observe that full satisfiability of a given CD can be established alternatively by checking independently class satisfiability and association satisfiability for every class and every association in the diagram. Intuitively, this claim is proven using the following argument: for every class C, there is an instance of the diagram in which the extension of C is nonempty. It can be shown (Maraee, 2007) that the (disjoint) union of all such instances is an instance of the diagram in which the extension of every class is nonempty. This argument holds due to the special character of UML CD constraints, which are invariant under disjoint instance union (Lenzerini and Nobili, 1990; Maraee and Balaban, 2006).

4.2 Reasoning methods

Considering the high complexity of industrial software, it can be very difficult to verify the properties of a UML CD and to guarantee that they are preserved during the designing phase. Thus, it is highly desirable to have CASE tools equipped with automated reasoning capabilities to support the designer. To this end, it is important to develop methods that can be implemented and integrated within such tools.

Reasoning methods for CDs should support inference of implicit consequences, improvement of the design, as well as handling problems of satisfiability and redundancy. Moreover, methods should be capable to *detect* problems, *identify* their source, and *suggest* how to resolve them. However, in practice most of the reasoning approaches addressed so far provide only problem detection. Yet, if one wants to have CASE tools that approach the level of current Integrated Development Environment compilers it is necessary to provide at least cause identification.

As developed so far, CDs reasoning methods can be classified into *concrete* reasoning methods that directly solve specific problems (Balaban and Maraee, 2006; Hartmann, 1995; Kaneiwa and Satoh, 2006; Lenzerini and Nobili, 1990), and into *translation-based* methods that support reasoning by mapping diagrams into a formal reasoning framework (Artale et al., 2007a; Berardi et al., 2005). Concrete methods are usually developed for detecting errors and revealing redundancy, while translation based methods can deal with more general inferences serving a variety of modeling needs.

In the translation-based methods, a UML CD is translated into a formula or expression in some formal language in such a way that the semantics of the diagram is fully captured, or at least the *soundness* and *completeness* of the reasoning is preserved. Once the CD is translated, one can rely on powerful inference mechanisms already implemented to verify the relevant properties of the diagram.

Before presenting in detail the existing methods, it is important to emphasize three essential features of such methods: *soundness*, *completeness* and *complexity*. Soundness means that the method gives only correct answers, while completeness means that the method can find always the correct answers. The complexity of reasoning refers to the efficiency of the method, which can be given in terms of the space and/or time required to find the answer. The methods described below demonstrate all these features.

Regarding translation based methods, we focus on those based in description logics. Although these methods implement DL with ExpTime reasoning problems, they are sound and complete (Artale et al., 2007a; Berardi et al., 2005).

Some concrete methods offer more efficient algorithms, but this generally leads to incomplete reasoning (due to the intrinsic complexity of reasoning in UML, see Section 4.3). Yet, other concrete methods attempt for efficient incomplete algorithms by resorting to finite reasoning.

4.2.1 Description logics based methods

Using description logics (DLs) is, probably, the most prominent approach for reasoning about CDs and databases using a general reasoner. It is motivated by the immediate similarity between the knowledge described by both formalisms. CDs describe a structured world of classes and associations, while

DLs describe a similar universe of concepts and roles. The ontological similarity between UML CDs and description logics, and the essential reasoning capabilities of description logic tools present DLs as natural candidates for supporting reasoning on CDs.

Exploiting the similarity between UML CDs and DLs, Berardi, Calvanese, and De Giacomo (2005) formalize CDs using very expressive description logics (Calvanese and De Giacomo, 2003). They show how UML CDs not including OCL constraints, but including completeness and disjointness constraints on generalization hierarchies, can be fully captured using the description logic $\mathcal{DLR}_{ifd}$. We present how the translation of UML to $\mathcal{DLR}_{ifd}$ is made, for each UML construct separately.

Classes

A class C corresponds to a $\mathcal{DLR}_{ifd}$ concept C. To capture an attribute a of type T for a class C, a binary relation a is used. The typing of the attribute is expressed by the assertion:

$$C \sqsubseteq \forall[1](a \Rightarrow (2 : T))$$

Such an assertion specifies that, for each instance c of the concept C, all objects related to c by a are instances of T. The following assertion is added to specify a multiplicity $[i..j]$ associated to the attribute:

$$C \sqsubseteq (\geq i[1]a) \sqcap (\leq j[1]a)$$

An operation $f(P_1, \ldots, P_m) : Q$ of a class is formalized as a $\mathcal{DLR}_{ifd}$ relation $f_{P_1,\ldots,P_m}$ of arity $1+m+1$ among instances of the $\mathcal{DLR}_{ifd}$ concepts $C, P_1, \ldots, P_m, Q$. To capture the semantics of the operation the following assertions are needed.

- Correct types of the parameters:

$$f_{P_1,\ldots,P_m} \sqsubseteq (2 : P_1) \sqcap \cdots \sqcap (m+1 : P_m)$$

- Functionality of the relation

$$(\textbf{fd}\ \ f_{P_1,\ldots,P_m}\, 1, \ldots, m+1 \rightarrow m+2)$$

- Correct typing of the result

$$C \sqsubseteq \forall[1](f_{P_1,\ldots,P_m} \Rightarrow (m+2 : Q))$$

Associations

An n-ary association R between the classes $C_1, \ldots, C_n$, without a related association class, is represented by an n-ary $\mathcal{DLR}_{ifd}$ relation R that satisfies the following typing assertion:

$$R \sqsubseteq (1 : C_1) \sqcap (2 : C_2) \sqcap \ldots \sqcap (n : C_n)$$

An n-ary association with a related association class C_R, on the other hand, is formalized in $\mathcal{DLR}_{ifd}$ by reifying the association into a $\mathcal{DLR}_{ifd}$ concept C_R with n binary relations $P_1, \ldots, P_n$, one for each component of the association. The semantics of the associations is enforced by the assertions:

$$\begin{array}{rcl} C_R & \sqsubseteq & \exists[1]R_1 \sqcap (\leq 1\,[1]R_1) \sqcap \forall[1](R_1 \Rightarrow (2 : C_1)) \sqcap \\ & & \exists[1]R_2 \sqcap (\leq 1\,[1]R_2) \sqcap \forall[1](R_2 \Rightarrow (2 : C_2)) \sqcap \\ & & \quad \vdots \\ & & \exists[1]R_n \sqcap (\leq 1\,[1]R_n) \sqcap \forall[1](R_n \Rightarrow (2 : C_n)) \end{array}$$

where $\exists[i]R_j$ (with $i \in \{1, \ldots, n\}$) specifies that the concept C_R must have all components of the association, $(\leq 1[1]R_i)$ specifies that each such component is single valued, $\forall[1]R_i \Rightarrow (2 : C_i)$ specifies to which class each component has to belong. Finally, in order to faithfully represent the association by a class the following identification assertion is needed

$$(\textbf{id}\ \ C_R\,[1]R_1, \ldots, [1]R_n)$$

such an assertion specifies that each instance of C_R represents a distinct tuple in $C_1 \times \ldots \times C_n$.

Multiplicities for binary associations with no related association class are formalized in $\mathcal{DLR}_{ifd}$ by the following assertions:

$$C_1 \sqsubseteq (\geq n_l[1]\,R) \sqcap (\leq n_u[1]\,R)$$

$$C_2 \sqsubseteq (\geq m_l[2]\,R^-) \sqcap (\leq [2]m_u\,R^-)$$

For associations with a related class, the multiplicity constraints can be imposed on the relations modeling the components (roles) of the associations. Since the names of such relations are unique with respect to the association only, and not with respect to the entire diagram, such constraints are stated in $\mathcal{DLR}_{ifd}$ as follows:

$$C_1 \sqsubseteq (\geq n_l[2](R_1 \sqcap (1 : C_R))) \sqcap (\leq n_u[2](R_1 \sqcap (1 : C_R)))$$

$$C_2 \sqsubseteq (\geq m_l[2](R_2 \sqcap (1 : C_R))) \sqcap (\leq m_u[2](R_2 \sqcap (1 : C_R)))$$

Generalizations

A UML class C generalizing a class C_1 is captured in $\mathcal{DLR}_{ifd}$ by the assertion $C_1 \sqsubseteq C$. The encoding in $\mathcal{DLR}_{ifd}$ also captures inheritance among association classes and multiple inheritance between classes. A class hierarchy conformed by a superclass C with the subclasses $C_1, \ldots C_n$ can be represented by the assertions

$$C_i \sqsubseteq C \qquad \text{for each } i \in \{1, \ldots n\}$$

Disjointness among the classes $C_1, \ldots, C_n$ is expressed by

$$C_i \sqsubseteq \prod_{j=i+1}^{n} \neg C_j \quad \text{for } i = 1, \ldots, n-1\,,$$

while a completeness constraint expressing that each instance of C is an instance of at least one of $C_1, \ldots, C_n$ is be given by

$$C \sqsubseteq \bigsqcup_{i=1}^{n} C_i\,.$$

The encoding presented above captures correctly the semantics of UML CDs. Indeed, there is a direct correspondence between instantiations of UML CDs and models of the corresponding $\mathcal{DLR}_{ifd}$ TBox. This is captured by the following theorem.

Theorem 4.5 (Berardi et al. 2005, Theorem 6.6). *Let $\mathcal{D}$ be a UML CD and $\mathcal{T}_\mathcal{D}$ the $\mathcal{DLR}_{ifd}$ TBox constructed as described above. Then every legal instantiation of $\mathcal{D}$ is a model of $\mathcal{T}_\mathcal{D}$, and vice-versa.*

A consequence of this result is that reasoning on UML CDs can be performed by reasoning on $\mathcal{DLR}_{ifd}$ knowledge bases. In particular, we have that class satisfiability on UML CDs can be reduced to concept satisfiability in $\mathcal{DLR}_{ifd}$.

Theorem 4.6 (Berardi et al. 2005, Theorem 6.7). *Let $\mathcal{D}$ be a UML CD and $\mathcal{T}_\mathcal{D}$ the TBox constructed as above. Then a class C is satisfiable in $\mathcal{D}$ if and only if the concept C is satisfiable w.r.t. $\mathcal{T}_\mathcal{D}$.*

This means that it is possible to exploit reasoning tools developed for DLs to reason on UML CDs. However, current state-of-the-art DL based reasoning systems do not support functional dependencies and identification constraints. The apparent problem that functional dependencies and identification constraints pose is tackled by resorting to the simpler description logic $\mathcal{ALCQI}$.

Indeed $\mathcal{ALCQI}$ does not include functional dependencies or identification constraints, which play a special role, since they allow to capture correctly the semantics of n-ary associations and of operations. However, due to the tree model property of $\mathcal{DLR}_{ifd}$, when one does not want to specifically reason about functional dependencies or identification constraints (which is the case for class satisfiability), it is possible to drop such constraints from $\mathcal{DLR}_{ifd}$, and still preserve soundness and completeness of reasoning on concepts and relations (Berardi et al., 2005). As $\mathcal{ALCQI}$ only has binary relations, a mechanism for handling n-ary relations is needed. $\mathcal{DLR}_{ifd}$ relations of arity $n > 2$ are translated through reification. One concept representing the tuples of the relation is introduced and n $\mathcal{ALCQI}$ (binary) functional roles, one for each component of the relation, are introduced. The tree model property guaranties that such translation is faithful, in the sense that there will be no two instances of the concept representing the same tuple of the relation (Calvanese et al., 2001b). Moreover, the encoding in $\mathcal{ALCQI}$ is correct in the sense that it preserves class satisfiability.

Theorem 4.7 (Berardi et al. 2005, Theorem 7.2). *Let $\mathcal{D}$ be a UML CD and $\mathcal{T}_\mathcal{D}$ the $\mathcal{ALCQI}$ TBox corresponding to $\mathcal{D}$. A class C is satisfiable in $\mathcal{D}$ if and only if the concept C is satisfiable w.r.t. $\mathcal{T}_\mathcal{D}$.*

Current state-of-the-art DL reasoning systems (Haarslev and Möller, 2001; Horrocks, 1998) support arbitrary complex $\mathcal{ALCQI}$ knowledge bases and implement sound and complete reasoning algorithms. These algorithms are based on tableaux techniques (Baader and Sattler, 2001; Horrocks et al., 1999) and, although not optimal from the computational complexity point of view, they are highly optimized and exhibit good average case performance (Horrocks, 2003).

4.2.2 Concrete methods

Kaneiwa and Satoh (2006) study the problem of full satisfiability on a restricted set of UML CDs that include classes with typed attributes and cardinality constraints on the attributes, unconstrained associations and constrained generalization sets. They describe three *triggers* for inconsistency in such diagrams and provide algorithms for deciding full satisfiability of the restricted CDs. These triggers are described as follows:

$\mathcal{T}_1$ **(generalization + disjointness).** The first trigger refers to the interaction of generalization and disjointness constraints. The inconsistency appears when a class C has a superclass C_k but the classes C and C_k are constrained to be disjoint in other class hierarchy. Figure 4.4 shows an inconsistency situation caused by generalization and disjointness constraints.

$\mathcal{T}_2$ **(overwriting/multiple inheritance).** The second inconsistency trigger refers to the following situations:

1. Possible conflicts between types/multiplicities in inherited attributes. Figure 4.5a illustrates a class C_2 that inherits the attribute $a : T_1[i..j]$ from class C_1, where the type and multiplicity of a are *overwritten* by $T_2[i'..j']$. The conflict occurs if T_1 and T_2 are incompatible types or if $i' > i$. In Figure 4.5b the attribute a is inherited to class C from two classes C_1 an C_2. This *multiple* inheritance can cause again a conflict between the types T_1 and T_2 or between the multiplicities $[i..j]$ and $[i'..j']$.
2. Possible conflicts between cardinality constraints over association generalizations. The inconsistency occurs when the refinements over cardinality constraints on associations are unrealizable.

$\mathcal{T}_3$ **(completeness + disjointness).** Interactions of disjointness and completeness constraints constitute the third inconsistency trigger. Interactions between these constraints can yield inconsistencies on a CD. In the simplest scenario we have that the diagram enforces covering of a class C by the classes $C_1, \ldots, C_n$ and disjointness between classes $C, C'_1, \ldots, C'_m$. The inconsistency occurs when $\{C_1, \ldots, C_n\} \subseteq \{C'_1, \ldots, C'_m\}$. Figure 4.6 shows an example of inconsistencies caused by the interactions of covering and disjointness constraints.

Based on these inconsistency patterns, Kaneiwa and Satoh provide tractable algorithms for deciding full satisfiability of restricted CDs. They provide a PSPACE upper bound for full satisfiability. However, the results in Chapter 5 show that the algorithms in (Kaneiwa and Satoh, 2006) must be incomplete.

The algorithms operate on an FOL encoding of a given UML CD, which serves as an indirect semantics definition. They analyze the structure of the given diagram and identify occurrences of the inconsistency triggers. Consequently, this concrete method functions as inconsistency detection algorithm and cause identification algorithm as well.

This pattern-based method for detecting unsatisfiability is implemented as a debugging system for restricted CDs (Satoh et al., 2006). The debugging is based on an elaborated set of rules for contradiction detection. The rules enable identification of the part in the UML diagram that cause the inconsistency, and suggest a possible solution.

4.2.3 Concrete methods for finite Reasoning

Reasoning on finiteness of entity relationship and CDs has attracted much attention. The problem was independently identified by Lenzerini and Nobili (1990) and by Thalheim (1992, 1993), and referred to entity relationship

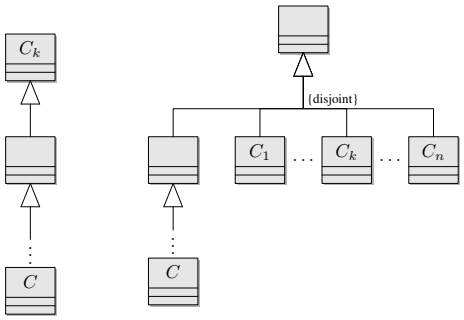
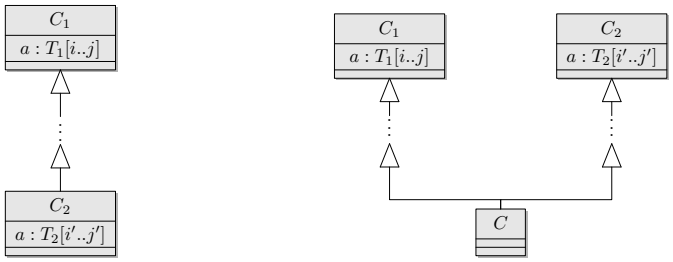


Figure 4.4: Inconsistency triggered by generalization and disjointness



(a) Overwriting inheritance (b) Multiple inheritance

Figure 4.5: Inconsistency triggered by multiple/overwriting inheritance

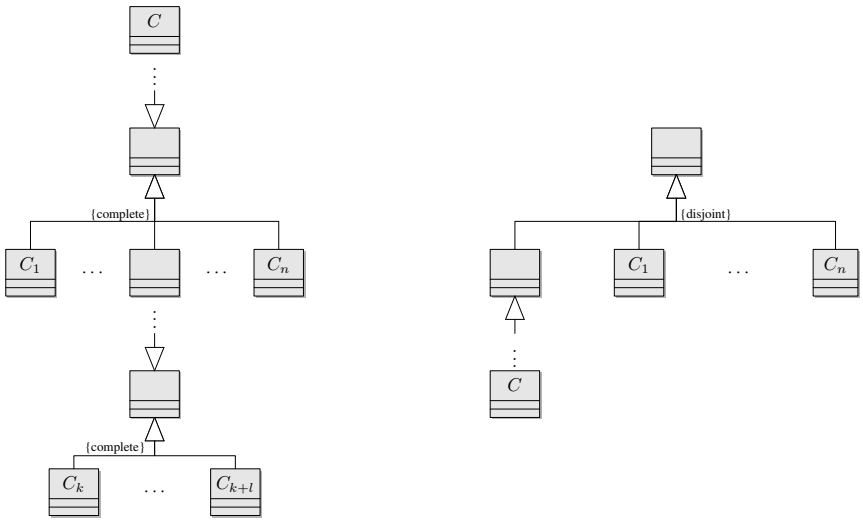


Figure 4.6: Inconsistency triggered by disjointness and covering

diagrams. Later on, the methods have been extended to various fragments of UML CDs. The problem in its simplest form consists in detecting whether a UML CD is not fully finitely satisfiable, i.e., whether there is a model such that the extension of every class is finite. However, it is also desirable to identify cause, and suggest repair in diagrams that are not fully finitely satisfiable.

There are two main approaches to this problem: the *linear programming approach* and the *graph based approach*. The first approach reduces the full finite satisfiability problem to the problem of finding a solution to a system of linear inequalities. The second approach detects infinity causing cycles in the diagram, and possibly suggest repair transformations. All methods apply only to fragments of UML CDs. Detection of infinity in unrestricted UML CDs is still an open issue.

Linear Programming Based Finite Satisfiability Detection

The fundamental method of Lenzerini and Nobili (1990) is defined for an entity relationship diagram that includes entity types (classes), n-ary relationship types (associations), and cardinality (multiplicity) constraints. The method consists of a transformation of the cardinality constraints into a set of linear inequalities whose variables stand for the cardinalities of the entity and relationship types in a possible instance.

The main 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 CDs can be decided in polynomial time.

Calvanese and Lenzerini (1994) extend the inequalities based method of Lenzerini and Nobili to apply to schemata with ISA (class hierarchy) constraints. The expansion is based on the assumption that class extensions may overlap. They provide a two stage algorithm in which the full finite satisfiability problem of a CD with ISA constraints is reduced into the full finite satisfiability problem of a CD having no class hierarchy constraints. Then, similarly to Lenzerini and Nobili (1990), the full finite satisfiability of the new CD is checked by testing whether a derived linear inequalities system has a solution. The linear inequalities system derived in Calvanese and Lenzerini (1994) is different from the one derived by Lenzerini and Nobili (1990). The former is quite complex, and might introduce, in the worst case, an exponential number of new classes and associations, in terms of the input diagram size. This method was simplified by Cadoli, Calvanese, De Giacomo, and Mancini (2004), by restricting class overlapping to class hierarchy alone. The latter simplification reduces the overall numbers of classes and association, but still introduces, in the worst case, an exponential number of new classes and associations.

Balaban and Maraee (2006, 2007) extend the inequalities based method of Lenzerini and Nobili to UML CDs with (1) binary association; (2) class hierarchy (ISA) constraints; (3) generalization set constraints disjoint/overlapping and complete/incomplete; (4) n-ary association with the standard interpretation of cardinality constraints; (5) qualifier dependent cardinality constraint; (6) association classes; (7) association hierarchy constraints. The extension is based on a preprocessing reduction of full finite satisfiability of a given CD, to the full finite satisfiability of restricted CDs handled by the Lenzerini and Nobili method.

The occurrences of generalization set constraints require the addition of new inequalities, and to restrict the method scope to class hierarchies whose undirected graph structure is acyclic. The method introduces only a linear number of new associations and inequalities (linear in the diagram size) and requires only a linear inequalities solver. Therefore it is simple to extend a UML CASE tool with this method (Maraee, 2007) . On the negative side, this method does not apply to cyclic class hierarchies.

The three linear inequalities based methods described above act only as detectors for full finite satisfiability.

4.3 Complexity results on reasoning

The ultimate goal for defining reasoning tasks on UML CDs is to integrate them within CASE tools in order to verify semantic properties of the diagrams in an automated fashion. For this reason, the implementation of reasoning methods should take into account the optimality of the algorithms used to make the inferences associated with each reasoning task. The study of optimal algorithms starts from the elicitation of the computational complexity of the problem the algorithm should solve. Knowing the intrinsic complexity of each reasoning task is therefore essential to avoid using too powerful inference mechanisms and to select the most appropriate mechanism to tackle each reasoning problem. In this section, we present complexity bounds for class satisfiability of UML CDs.

The computational complexity of class satisfiability of UML CDs including all constructs and constraints has not been studied. However, as OCL constraints are essentially full first order logic formulas, they would make reasoning undecidable. Regarding decidable reasoning, Berardi et al. (2005) have established the ExpTime-completeness of class satisfiability of UML CDs under fairly restrictive assumptions, namely: only binary associations, only minimal multiplicity constraints, generalizations (over classes and associations) with disjointness and completeness constraints. In order to avoid intractable reasoning, various constraint combinations have been considered. Artale et al. (2007a) address class satisfiability in three sublan-

guages of UML[1]. The first one allows for unconstrained generalization between associations, and completeness and disjointness constraints on class hierarchies only. The second, forbids generalization between associations, and the third one, restrict further by disallowing completeness constraints over class generalizations.

In this section we will present the complexity bounds for UML CDs that include at least the following constructs and constraints:

- classes
- binary associations
- generalization
- disjointness and completeness constraints over generalizations
- refinement of multiplicity constraints over generalization

It is worth to mention that the presence of attributes does not affect the complexity results that we will discuss here, and therefore, they are not taken into account in our classification. The following table summarizes each one of these languages:

Language	**Constraints**					
	Classes			Associations		
	ISA	disjointness	completeness	ISA	multiplicity	refinement
UML_{isaR}	✓	✓	✓	✓	✓	✓
UML_{Bool}	✓	✓	✓	✗	✓	✓
UML_{ref}	✓	✓	✗	✗	✓	✓

4.3.1 Upper complexity bounds for class satisfiability

A consequence of Theorem 4.5 is that reasoning on UML CDs can be performed by reasoning on $\mathcal{DLR}_{ifd}$ knowledge bases. Since class satisfiability can be reduced to concept satisfiability in $\mathcal{DLR}_{ifd}$ (see Theorem 4.6) and reasoning in $\mathcal{DLR}_{ifd}$ is ExpTime-complete (Calvanese et al., 2001b), we get an ExpTime upper bound for class satisfiability of UML CDs.

Tighter upper complexity bounds have been established for class satisfiability in the sublanguages: UML_{isaR}, UML_{Bool} and UML_{ref}. Again by exploiting the correspondence between conceptual modeling formalisms and variants of description logics, Artale et al. (2007a) strength the results of

[1] The results in (Artale et al., 2007a) are formulated in terms of the Entity-Relationship model, but they carry directly over also to UML class diagrams.

Berardi, Calvanese, and De Giacomo (2005), by showing that even when dropping disjointness and completeness constraints over association generalizations class satisfiability stays in ExpTime in UML_{isaR}.

Class satisfiability of UML_{Bool} diagrams, on the other hand, has been shown to be in NP. The result follows by encoding UML_{Bool} CDs into DL-Lite$_{Bool}$ KBs in such a way that the size of the DL-Lite$_{Bool}$ TBox corresponding to the diagram is polynomial in the size of the diagram. The translation preserves soundness and completeness w.r.t. class satisfiability. The NP upper complexity bound holds because reasoning in DL-Lite$_{Bool}$ is known to be NP-complete (Artale et al., 2007b).

By restricting to UML_{ref} diagrams, the complexity of class satisfiability decreases even more. UML_{ref} diagrams can be encoded as DL-Lite$_{krom}$ knowledge bases which results in a NLogSpace upper bound for class satisfiability in UML_{ref}, as reasoning in DL-Lite$_{krom}$ is known to be NLogSpace (Artale et al., 2007b).

4.3.2 Lower complexity bounds for class satisfiability

Class satisfiability of UML CDs without OCL constraints but completeness and disjointness constraints over class and association hierarchies has been shown to be ExpTime-hard. This result follows from a reduction of atomic concept satisfiability w.r.t. $\mathcal{ALC}^-$ KBs to class satisfiability in UML (Berardi et al., 2005). The result carries over since atomic concept satisfiability in $\mathcal{ALC}^-$ is ExpTime-hard (Berardi et al., 2005). Due to its simplicity, $\mathcal{ALC}^-$ TBoxes can be easily encoded into UML_{isaR} diagrams: atomic concepts are directly encoded as classes. The following encoding is used:

- Each atomic concept A is encoded as a class A in $\mathcal{D}$.
- A class O that generalizes (possibly indirectly) all classes in $\mathcal{D}$ is introduced.

The assertions in the primitive $\mathcal{ALC}^-$ TBox $\mathcal{T}$ are encoded in the CD as follows:

1. For each assertion of the form $A \sqsubseteq B$, a generalization between the classes A and B is introduced.
2. For each assertion of the form $A \sqsubseteq \neg B$, the hierarchy shown in Figure 4.7 is constructed.
3. For each assertion of the form $A \sqsubseteq B_1 \sqcup B_2$, an auxiliary class B is introduced, and the subdiagram in Figure 4.8 is constructed.
4. For each assertion of the form $A \sqsubseteq \forall P.\, B$, a new class $\overline{A}$ and two new binary associations P_A and $P_{\overline{A}}$ are introduced, and the subdiagram

in Figure 4.9 is constructed. A and $\overline{A}$ are disjoint and there is a generalization with completeness constraint between P and its children P_A and $P_{\overline{A}}$. The intention of this encoding, is to enforce that each instance of A participating to P is in fact participating to P_A, and hence associated via P to an instance of B.

5. For each assertion of the form $A \sqsubseteq \exists P.\, B$, a new binary association P_{AB} is introduced, and the portion of diagram shown in Figure 4.10 is constructed. The multiplicity constraint $1..*$ enforces that for each instance of A there is an instance of B related to it through P_{AB}, and hence through P.

The encoding is such that the following result holds.

Lemma 4.8. *An atomic concept A is satisfiable w.r.t. an $\mathcal{ALC}^-$ TBox $\mathcal{T}$ if and only if the class A is satisfiable in the UML CD $\mathcal{D}$ constructed as above.*

The ExpTime-hardness of atomic concept satisfiability in $\mathcal{ALC}^-$ (Berardi et al., 2005) and the fact that the size of $\mathcal{D}$ is linear in the size of $\mathcal{T}$ imply the ExpTime-hardness of class satisfiability of UML CDs.

Class satisfiability in UML_{isaR}

Interestingly, by eliminating completeness and disjointness constraints over association hierarchies the complexity of class satisfiability does not decrease. It is still possible to encode an $\mathcal{ALC}^-$ TBox in UML_{isaR} in such a way that a similar result as the one in Lemma 4.8 holds (Artale et al., 2007a). To achieve this encoding, axioms of the form $A \sqsubseteq \forall P.\, B$ are encoded by reifying the binary relation P with a class C_P and adding two functional associations P_1 and P_2 that represent respectively the first and second component of the relation. A similar encoding is used to capture axioms of the form $A \sqsubseteq \exists P.\, B$ (see Figures 4.11 and 4.12).

Class satisfiability in UML_{Bool}

It turns out that the major source of complexity are the generalizations on associations. Class satisfiability in UML_{Bool} (where no hierarchy constraints on associations are allowed) is shown to be NP-hard (Artale et al., 2007a). This result follows from a reduction of the 3Sat problem, which is known to be NP-complete (Papadimitriou, 1994) to class satisfiability in UML_{Bool}.

Class satisfiability in UML_{ref}

The complexity of class satisfiability drops even more when restricting the expressiveness of the diagrams by disallowing completeness constraints on

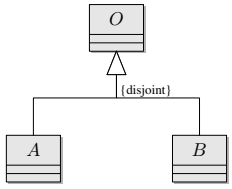


Figure 4.7: Encoding of the assertion $A \sqsubseteq \neg B$

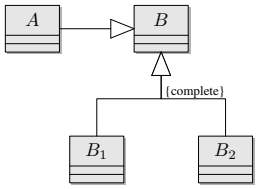


Figure 4.8: Encoding of the assertion $A \sqsubseteq B_1 \sqcup B_2$

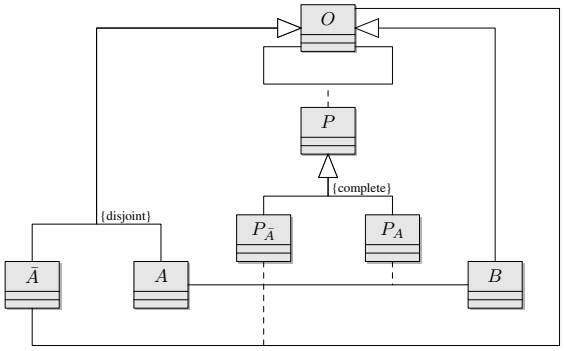


Figure 4.9: Encoding of the assertion $A \sqsubseteq \forall P.\, B$

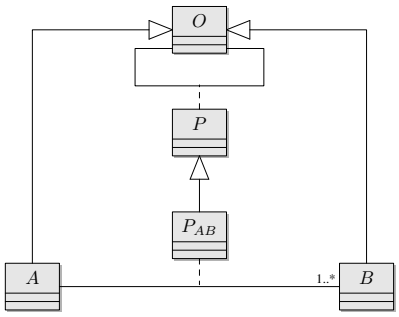


Figure 4.10: Encoding of $A \sqsubseteq \exists P.\, B$

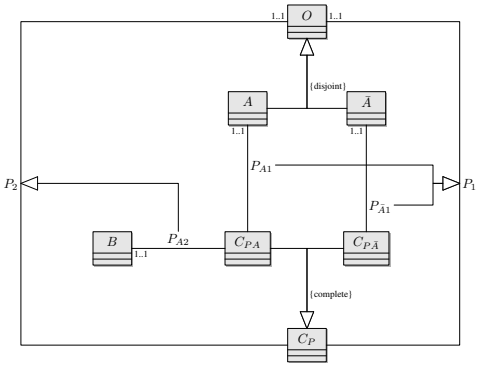


Figure 4.11: $A \sqsubseteq \forall P.\, B$ in UML_{isaR}

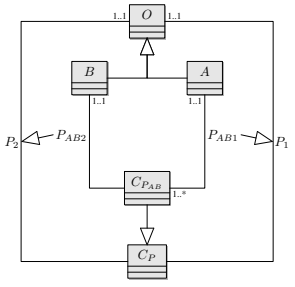


Figure 4.12: $A \sqsubseteq \exists P.\, B$ in UML_{isaR}

class hierarchies. Here, the reachability problem in directed graphs, which is known to be in NLogSpace (Papadimitriou, 1994) is reduced to class unsatisfiability in UML_{ref}.

Chapter 5

Complexity of full satisfiability

Checking full satisfiability of UML class diagrams can be regarded as a strategy for detecting *superfluous* (unsatisfiable) classes or associations in the diagram (i.e., classes/associations that are empty in every legal instantiation of the diagram). Detecting such superfluous elements can be seen as a helpful strategy for fixing inconsistencies in the diagrams. Resolving inconsistencies usually means correcting or removing one or more constraints. If no additional information is available, any constraint is a candidate for this. However, most constraints do not have any impact on the inconsistency. Thus, localizing the conflict is essential for fixing it.

In order to develop efficient methods for checking full satisfiability, it is important to determine first the intrinsic complexity of the problem. To the best of our knowledge, the complexity of full satisfiability of UML CDs has not been so far explicitly addressed. Indeed, the work of Kaneiwa and Satoh (2006) (described in Section 4.2.2) refers algorithms that could provide upper bounds for full satisfiability of restricted UML CDs. However, according to the results we present here, the PSPACE upper bound of the algorithm in (Kaneiwa and Satoh, 2006) for full satisfiability in a UML language that contains UML_{isaR} is not accurate. This suggests that such algorithm must be incomplete.

We present both upper and lower complexity bounds for full satisfiability of UML_{isaR}, UML_{Bool} and UML_{ref} CDs. In fact, our results show that the complexity of full satisfiability coincides in all cases with that of class satisfiability. The upper bounds are an almost direct consequence of the corresponding upper bounds of class satisfiability, and the disjoint union model property of UML CDs. To achieve these results, we exploit the formalization of UML CDs in terms of description logics, as described in (Berardi et al., 2005) (see also Section 4.2.1). The lower bounds, on the other hand, involve a careful adaptation of the corresponding proofs for class satisfiability.

One way to determine the complexity of full satisfiability in UML CDs, is to find a problem that can be easily reduced to it. The reduction of concept satisfiability in $\mathcal{ALC}$ to class satisfiability in UML, suggests that, by defining a comparable problem in $\mathcal{ALC}$ to that of full satisfiability of UML CDs would lead to a similar reduction. The only gap to fill in this scenario would be to determine the complexity of the problem in $\mathcal{ALC}$.

5.1 Full satisfiability in $\mathcal{ALC}$

We introduce a notion of TBox satisfiability that requires not only the existence of any model of a TBox, but also, that every atomic concept introduced in the terminology is meaningful. That means, a model in which the extension of every atomic concept is nonempty. Formally, we have the following definition.

Definition 5.1. Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox. $\mathcal{T}$ is said to be *fully satisfiable* if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $A^{\mathcal{I}} \neq \emptyset$, for every atomic concept A occurring in $\mathcal{T}$. ◇

Lemma 5.2. *Concept satisfiability w.r.t. an $\mathcal{ALC}$ TBox can be linearly reduced to TBox full satisfiability in $\mathcal{ALC}$.*

Proof: Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox and C an $\mathcal{ALC}$ concept. As pointed out in (Berardi et al., 2005), C is satisfiable w.r.t. $\mathcal{T}$ if and only if $C \sqcap A_{\mathcal{T}}$ is satisfiable w.r.t. the TBox $\mathcal{T}_1$ consisting of the single assertion

$$A_{\mathcal{T}} \sqsubseteq \bigsqcap_{C_1 \sqsubseteq C_2 \in \mathcal{T}} \neg C_1 \sqcup C_2 \sqcap \bigsqcap_{1 \leq i \leq n} \forall P_i . A_{\mathcal{T}}$$

where $A_{\mathcal{T}}$ is a fresh atomic concept and $P_1, \ldots, P_n$ are all the atomic roles occurring in $\mathcal{T}$ and C.

In order to reduce this problem to full satisfiability, we extend $\mathcal{T}_1$ to the TBox

$$\mathcal{T}_2 = \mathcal{T}_1 \cup \{A_C \sqsubseteq A_{\mathcal{T}} \sqcap C\} ,$$

with A_C a fresh atomic concept. We then prove that the following claim holds.

Claim. *$C \sqcap A_{\mathcal{T}}$ is satisfiable w.r.t. $\mathcal{T}_1$ iff $\mathcal{T}_2$ is fully satisfiable.*

Proof:

($\Rightarrow$) Let $\mathcal{I}$ be a model of $\mathcal{T}_1$ such that $(A_\mathcal{T} \sqcap C)^\mathcal{I} \neq \emptyset$. We construct a full model of $\mathcal{T}_2$. Let $\mathcal{J} = (\Delta^\mathcal{I} \cup \{d^{top}\}, \cdot^\mathcal{J})$, with $d^{top} \notin \Delta^\mathcal{I}$, we define

$$\begin{aligned} A_\mathcal{T}^\mathcal{J} &= A_\mathcal{T}^\mathcal{I}, \\ A_C^\mathcal{J} &= (A_\mathcal{T} \sqcap C)^\mathcal{I}, \\ A^\mathcal{J} &= A^\mathcal{I} \cup \{d^{top}\} \quad \text{for all atomic concepts } A \text{ in } \mathcal{T} \text{ and } C, \\ P^\mathcal{J} &= P^\mathcal{I} \quad \text{for all atomic roles.} \end{aligned}$$

Clearly, the extension of every atomic concept is nonempty in $\mathcal{J}$. Intuitively, the constraints imposed by the original TBox $\mathcal{T}$ are captured in $\mathcal{T}_1$ and in $\mathcal{T}_2$ in such a way that the atomic concept $A_\mathcal{T}$ stands for the elements of the domain that satisfy the constraints imposed by the original TBox. We then fully satisfy the TBox by stating that d^{top} is an instance of every atomic concept occurring in the original TBox $\mathcal{T}$ and in C.

Formally, to prove that $\mathcal{J}$ is indeed a model of $\mathcal{T}_2$, we show first that the following claim holds.

Claim. *Let subconcepts($\mathcal{T}_2$) denote the set of all the subconcepts occurring in $\mathcal{T}_2$. For all $D \in$ subconcepts($\mathcal{T}_2$), $D^\mathcal{I} \subseteq D^\mathcal{J}$.*

Proof: We can assume, without loss of generality, that D is in negation normal form, and proof the claim by induction on the structure of D.

Basis of induction:

- $D = A_\mathcal{T}$ and $D = A_C$. Follow from the construction.
- $D = A$. The statement follows from the definition of $\mathcal{J}$, as we have that
$$A^\mathcal{I} \subseteq A^\mathcal{I} \cup \{d^{top}\} = A^\mathcal{J}$$
- $D = \neg A$. Let $d \in (\neg A)^\mathcal{I}$. We have that $d \in \Delta^\mathcal{I} \setminus A^\mathcal{I}$. Then, as $d \neq d^{top}$, we can conclude that $d \in \Delta^\mathcal{J} \setminus A^\mathcal{J}$ and therefore, $d \in (\neg A)^\mathcal{J}$.

Induction hypothesis: Assume that the statement holds for the concepts C_1 and C_2.

Induction step:

- $D = C_1 \sqcap C_2$. Let $d \in (C_1 \sqcap C_2)^\mathcal{I}$. We have then by definition, that $d \in C_1^\mathcal{I}$ and $d \in C_2^\mathcal{I}$. By induction hypothesis, $d \in C_1^\mathcal{J}$ and $d \in C_2^\mathcal{J}$, this yields that $d \in (C_1 \sqcap C_2)^\mathcal{J}$, as required.

- $D = C_1 \sqcup C_2$. The proof is analogous as for the previous case.

- $D = \forall P.\, C_1$. Let $d \in (\forall P.\, C_1)^{\mathcal{I}}$, and let $(d, d') \in P^{\mathcal{J}}$. From the definition of $\mathcal{J}$, we have that $(d, d') \in P^{\mathcal{I}}$ and therefore $d' \in C_1^{\mathcal{I}}$. Then, by induction hypothesis $d' \in C_1^{\mathcal{J}}$. Hence, we can conclude $d \in (\forall P.\, C_1)^{\mathcal{J}}$.

- $D = \exists P.\, C_1$. Let $d \in (\exists P.\, C_1)^{\mathcal{I}}$. Then, there is a $d' \in \Delta^{\mathcal{I}}$, such that $(d, d') \in P^{\mathcal{I}}$ and $d' \in C_1^{\mathcal{I}}$. From the definition of $\mathcal{I}$, we have that $(d, d') \in P^{\mathcal{J}}$, and by induction hypothesis $d' \in C_1^{\mathcal{J}}$. Hence, we can conclude that $d \in (\exists P.\, C_1)^{\mathcal{J}}$.

Now, it is easy to show that $\mathcal{J}$ indeed satisfies every assertion in $\mathcal{T}_2$:

$$\begin{aligned} A_{\mathcal{T}}^{\mathcal{J}} = A_{\mathcal{T}}^{\mathcal{I}} &\subseteq \Big(\prod_{C_1 \sqsubseteq C_2 \in \mathcal{T}} \neg C_1 \sqcup C_2 \sqcap \prod_{1 \le i \le n} \forall P_i.\, A_{\mathcal{T}}\Big)^{\mathcal{I}} \\ &\subseteq \Big(\prod_{C_1 \sqsubseteq C_2 \in \mathcal{T}} \neg C_1 \sqcup C_2 \sqcap \prod_{1 \le i \le n} \forall P_i.\, A_{\mathcal{T}}\Big)^{\mathcal{J}} \\ A_C^{\mathcal{J}} = (A_{\mathcal{T}} \sqcap C)^{\mathcal{I}} &\subseteq (A_{\mathcal{T}} \sqcap C)^{\mathcal{J}} \end{aligned}$$

($\Leftarrow$) Conversely, every full model $\mathcal{J}$ of $\mathcal{T}_2$ is also a model of $\mathcal{T}_1$ with $(A_{\mathcal{T}} \sqcap C)^{\mathcal{J}} \neq \emptyset$, as $A_C^{\mathcal{J}} \subseteq (A_{\mathcal{T}} \sqcap C)^{\mathcal{J}}$. □

Theorem 5.3. *TBox full satisfiability in $\mathcal{ALC}$ is* ExpTime*-complete*

Proof: The ExpTime membership is straightforward, as deciding full satisfiability of a given $\mathcal{ALC}$ TBox $\mathcal{T}$ can be reduced to deciding satisfiability of the following TBox:

$$\mathcal{T} \cup \bigcup_{1 \le i \le n} \{\top \sqsubseteq \exists P'.\, A_i\}\,,$$

where $\{A_1, \ldots, A_n\}$ are all the atomic concepts occurring in $\mathcal{T}$, and the P' is a new atomic role.

The ExpTime-hardness follows from Lemma 5.2. □

We now modify the reduction from Lemma 5.2 so it applies also to primitive $\mathcal{ALC}^-$ TBoxes.

Theorem 5.4. *Full satisfiability of primitive $\mathcal{ALC}^-$ TBoxes is* ExpTime*-complete.*

Proof: The ExpTime membership follows from Theorem 5.3.

For proving the ExpTime-hardness, we reduce atomic concept satisfiability w.r.t. an $\mathcal{ALC}^-$ TBox, which is known to be ExpTime-hard (Berardi et al., 2005) to full satisfiability in $\mathcal{ALC}^-$.

Let $\mathcal{T}^- = \{A_j \sqsubseteq D_j \mid 1 \leq j \leq m\}$ be a primitive $\mathcal{ALC}^-$ TBox, and A_0 an atomic concept. By Lemma 5.2, we have that A_0 is satisfiable w.r.t. $\mathcal{T}^-$ if and only if the TBox $\mathcal{T}_2'$ containing the following axioms is fully satisfiable,

$$\begin{aligned} A_{\mathcal{T}^-} &\sqsubseteq \bigsqcap_{A_j \sqsubseteq D_j \in \mathcal{T}^-} \neg A_j \sqcup D_j \sqcap \bigsqcap_{1 \leq i \leq n} \forall P_i.\, A_{\mathcal{T}^-} \\ A_0' &\sqsubseteq A_{\mathcal{T}^-} \sqcap A_0\,. \end{aligned}$$

where $A_{\mathcal{T}^-}$, A_0' fresh atomic concepts. $\mathcal{T}_2'$ is not a primitive $\mathcal{ALC}^-$ TBox, but it can be transformed into one preserving full satisfiability. First note that $\mathcal{T}_2'$ is equivalent to:

$$\begin{aligned} A_{\mathcal{T}^-} &\sqsubseteq \neg A_1 \sqcup D_1 \\ &\vdots \\ A_{\mathcal{T}^-} &\sqsubseteq \neg A_m \sqcup D_m \\ A_{\mathcal{T}^-} &\sqsubseteq \forall P_1.\, A_{\mathcal{T}^-} \\ &\vdots \\ A_{\mathcal{T}^-} &\sqsubseteq \forall P_n.\, A_{\mathcal{T}^-} \\ A_0' &\sqsubseteq A_{\mathcal{T}^-} \sqcap A_0 \end{aligned}$$

To get a primitive $\mathcal{ALC}^-$ TBox $\mathcal{T}_2^-$ we replace each axiom of the form $A_{\mathcal{T}^-} \sqsubseteq \neg A_j \sqcup D_j$ by the axioms

$$A_{\mathcal{T}^-} \sqsubseteq B_j^1 \sqcup B_j^2$$

$$B_j^1 \sqsubseteq \neg A_j$$

$$B_j^2 \sqsubseteq D_j$$

Claim. *$\mathcal{T}_2'$ is fully satisfiable if and only if $\mathcal{T}_2^-$ is fully satisfiable.*

Proof:

($\Rightarrow$) Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a full model of $\mathcal{T}_2'$. We can extend $\mathcal{I}$ into a full model $\mathcal{J}$ of $\mathcal{T}_2^-$. Let $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}} \cup \{d^+, d^-\}$, with $\{d^+, d^-\} \cap \Delta^{\mathcal{I}} = \emptyset$, and

define $\cdot^{\mathcal{J}}$ as follows:

$$\begin{aligned}
A_{\mathcal{T}}^{\mathcal{J}} &= A_{\mathcal{T}}^{\mathcal{I}}, \\
(A_0')^{\mathcal{J}} &= (A_0')^{\mathcal{I}}, \\
A^{\mathcal{J}} &= A^{\mathcal{I}} \cup \{d^+\} \quad \text{for every other atomic concept in } \mathcal{T}_2', \\
(B_j^1)^{\mathcal{J}} &= (\neg A)^{\mathcal{J}}, \text{ and} \\
(B_j^2)^{\mathcal{J}} &= (D_j)^{\mathcal{J}}, \text{ for each } A_{\mathcal{T}^-} \sqsubseteq B_j^1 \sqcup B_j^2 \in \mathcal{T}_2^-, \\
P^{\mathcal{J}} &= P^{\mathcal{I}} \cup \{(d^+, d^+)\} \quad \text{for every atomic role } P \text{ in } \mathcal{T}_2'
\end{aligned}$$

It is easy to see now, that $\mathcal{J}$ satisfies every axiom in $\mathcal{T}_2^-$.

($\Leftarrow$) The implication in this direction holds trivially, as every model of $\mathcal{T}_2^-$ is also a model of $\mathcal{T}_2'$. □

5.2 Full satisfiability of UML CDs

In this section, we use the results presented in the previous section to establish the hardness of full satisfiability of UML CDs. We reduce full satisfiability of primitive $\mathcal{ALC}^-$ TBoxes to full satisfiability of *restricted* UML CDs. The reduction requires at least the expressiveness of UML_{isaR} CDs. For less expressive UML languages, we provide suitable reductions to problems of already known complexity.

5.2.1 Full satisfiability in UML_{isaR}

We reduce full satisfiability of primitive $\mathcal{ALC}^-$ TBoxes to full satisfiability of UML_{isaR} CDs as follows.

Given a primitive $\mathcal{ALC}$ TBox $\mathcal{T}$, we construct a UML CD $\Sigma(\mathcal{T})$. For each atomic concept A in $\mathcal{T}$, we introduce a class A in $\Sigma(\mathcal{T})$. Additionally, we introduce a class O that generalizes (possibly indirectly) all the classes in $\Sigma(\mathcal{T})$ that represent an atomic concept. For each atomic role P, we introduce a class C_P which reifies the binary relation P. Moreover, we introduce two functional associations P_1, and P_2 that respectively represent the first and second components of the reified relation P.

The assertions in the primitive $\mathcal{ALC}^-$ TBox $\mathcal{T}$ are encoded in the CD as follows:

1. Assertions of the form $A \sqsubseteq B$, $A \sqsubseteq \neg B$, and $A \sqsubseteq B_1 \sqcup B_2$, are handled in the same way as the reduction presented in Section 4.3.2. For the sake of clarity, we present again the diagrams in Figure 5.1.

2. For each assertion of the form $A \sqsubseteq \forall P.\, B$, we add the auxiliary classes C_{PA_+} and $C_{P\bar{A}}$, and the associations $P_{\bar{A}}$, $P_{A_{+1}}$ and $P_{A_{+2}}$, and construct the diagram shown in Figure 5.2.

3. For each assertion of the form $A \sqsubseteq \exists P.\, B$, we add the auxiliary class $C_{P_{AB}}$, and construct the diagram shown in Figure 5.3.

We need a special treatment of the assertions $A \sqsubseteq \forall P.\, B$. We cannot use the encoding presented in Section 4.3.2 because UML_{isaR} does not allow for disjointness and completeness constraints over association generalizations. For this reason, we follow the approach taken in (Artale et al., 2007a), where this kind of assertions are encoded by reifying P with a class C_P and two functional associations P_1 and P_2. However, the encoding needs further modifications, as there is no guarantee that all relations in $\mathcal{T}$ are satisfiable. For example, if $A \sqsubseteq \forall P.\, \bot$ is a consequence of $\mathcal{T}$, then we cannot represent directly with a class the tuples with a first component in A. Instead, we represent a superset of them using the class C_{PA_+}. The encoding captures correctly full satisfiability, as stated in the following lemma.

Lemma 5.5. *A primitive $\mathcal{ALC}$ TBox $\mathcal{T}$ is fully satisfiable if and only if the UML CD $\Sigma(\mathcal{T})$ is fully satisfiable.*

Proof:

($\Leftarrow$) Let $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ be a legal instantiation of $\Sigma(\mathcal{T})$, such that the extension of every class is nonempty. We construct a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{T}$ by taking

$$\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}, \qquad A^{\mathcal{I}} = A^{\mathcal{J}}, \qquad P^{\mathcal{I}} = (P_1^- \circ P_2)^{\mathcal{J}},$$

for all atomic concepts A, and all atomic roles P in $\mathcal{T}$.

Clearly, $A^{\mathcal{J}} \neq \emptyset$. Now, we show that $\mathcal{I}$ indeed satisfies every assertion in $\mathcal{T}$. We analyze each case.

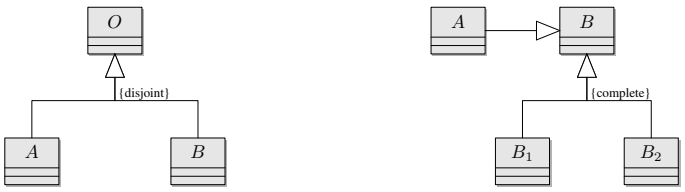
1. For each assertion of the form $A \sqsubseteq B$ in $\mathcal{T}$, there is a generalization in $\Sigma(\mathcal{T})$, such that $\mathcal{J}$ is a model of the FOL sentence $\forall x.\, (A(x) \rightarrow B(x))$.

   Hence, $\mathcal{J}$ assigns extensions to A and B in such a way that $A^{\mathcal{J}} \subseteq B^{\mathcal{J}}$, which by the definition of $\mathcal{I}$ implies that $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$.

2. For each assertion of the form $A \sqsubseteq \neg B$ in $\mathcal{T}$, we have the hierarchy shown in Figure 4.7, characterized by a disjointness constraint between the classes A and B. $\mathcal{J}$ assigns to the classes A, B and O the sets $A^{\mathcal{J}}$, $B^{\mathcal{J}}$ and $O^{\mathcal{J}}$, respectively, in such a way that

   $$A^{\mathcal{J}} \subseteq O^{\mathcal{J}}, \qquad B^{\mathcal{J}} \subseteq O^{\mathcal{J}} \qquad \text{and} \qquad A^{\mathcal{J}} \cap B^{\mathcal{J}} = \emptyset.$$

   From the latter, we have that $A^{\mathcal{J}} = \Delta^{\mathcal{J}} \setminus B^{\mathcal{J}}$. Then, by the definition of $\mathcal{I}$, we obtain that $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}} = (\neg B)^{\mathcal{I}}$.



(a) Encoding of $A \sqsubseteq \neg B$ (b) Encoding of $A \sqsubseteq B_1 \sqcup B_2$

Figure 5.1: Reduction of $\mathcal{ALC}^-$ to UML_{isaR}

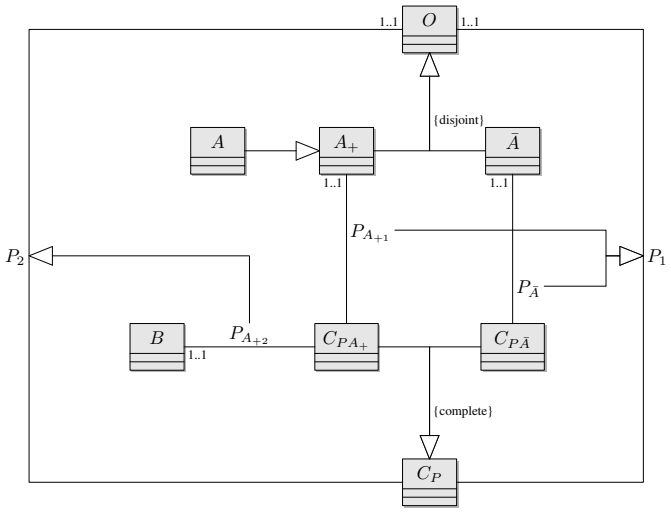


Figure 5.2: Encoding of the assertion $A \sqsubseteq \forall P.\, B$

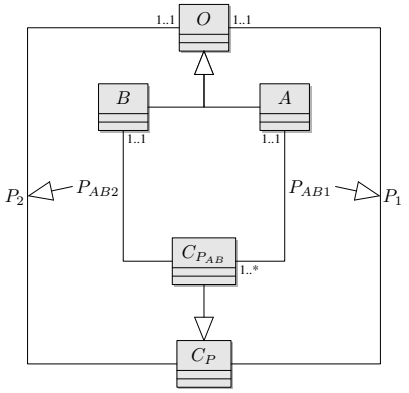


Figure 5.3: Encoding of $A \sqsubseteq \exists P.\, B$

3. For each assertion of the form $A \sqsubseteq B_1 \sqcup B_2$ in $\mathcal{T}$, we have that $\mathcal{J}$ assigns extensions to the classes A, B, B_1, and B_2, in such a way that

$$A^{\mathcal{J}} \subseteq B^{\mathcal{J}} \quad \text{and} \quad B^{\mathcal{J}} = B_1^{\mathcal{J}} \cup B_2^{\mathcal{J}},$$

which, again by the construction of $\mathcal{I}$, implies that $A^{\mathcal{I}} \subseteq B_1^{\mathcal{I}} \cup B_2^{\mathcal{I}} = (B_1 \sqcup B_2)^{\mathcal{I}}$.

4. For assertions of the form $A \sqsubseteq \forall P.B$, we have to show that

$$A^{\mathcal{I}} \subseteq (\forall P.B)^{\mathcal{I}}.$$

Let $o \in A^{\mathcal{I}} = A^{\mathcal{J}}$ and $o' \in \Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$, such that $(o, o') \in P^{\mathcal{I}}$. We have to show that $o' \in B^{\mathcal{I}}$, and hence that $o \in (\forall P.B)^{\mathcal{I}}$.

As $P^{\mathcal{I}} = (P_1^- \circ P_2)^{\mathcal{J}}$, there is an element $o'' \in \Delta^{\mathcal{J}}$ such that $(o, o'') \in (P_1^-)^{\mathcal{J}}$, and $(o'', o') \in P_2{}^{\mathcal{J}}$. Then, $o'' \in C_P^{\mathcal{J}}$, and by the completeness constraint, $o'' \in C_{PA_+}^{\mathcal{J}} \cup C_{P\bar{A}}^{\mathcal{J}}$.

We claim that $o'' \in C_{PA_+}^{\mathcal{J}}$. Let us show this by contradiction. Suppose that $o'' \in C_{P\bar{A}}^{\mathcal{J}}$. Then, there is a unique $a \in \Delta^{\mathcal{J}}$, such that $(o'', a) \in P_{\bar{A}}^{\mathcal{J}}$ and $a \in \bar{A}^{\mathcal{J}}$. It follows from $P_{\bar{A}}^{\mathcal{J}} \subseteq P_1^{\mathcal{J}}$ and by the multiplicity constraint over C_P, that $a = o$. This rises a contradiction, because $o \in A^{\mathcal{J}} \subseteq A_+^{\mathcal{J}}$ and, $A_+^{\mathcal{J}}$ and $\bar{A}^{\mathcal{J}}$ are disjoint.

Since $o'' \in C_{PA_+}^{\mathcal{J}}$, there is a unique $b \in \Delta^{\mathcal{J}}$ with $(o'', b) \in P_{A_{+2}}^{\mathcal{J}}$ and $b \in B^{\mathcal{J}}$. From $P_{A_{+2}}^{\mathcal{J}} \subseteq P_2^{\mathcal{J}}$ and the multiplicity constraint on C_P, it follows that $b = o'$. Thus, we have that $o' \in B^{\mathcal{J}} = B^{\mathcal{I}}$, and therefore, $o \in (\forall P.B)^{\mathcal{I}}$.

5. For each assertion of the form $A \sqsubseteq \exists P.B$ in $\mathcal{T}$, we have to show that

$$A^{\mathcal{I}} \subseteq (\exists P.B)^{\mathcal{I}}.$$

Let $o \in A^{\mathcal{I}} = A^{\mathcal{J}}$. Then, there is an element $o' \in \Delta^{\mathcal{J}}$ such that $(o', o) \in (P_{AB1})^{\mathcal{J}}$ and $o' \in C_{P_{AB}}$.

Since $o' \in C_{P_{AB}}^{\mathcal{J}}$, there is $o'' \in \Delta^{\mathcal{J}}$ with $(o', o'') \in P_{AB2}^{\mathcal{J}}$ and $o'' \in B^{\mathcal{J}} = B^{\mathcal{I}}$.

Then, as $P_{AB2}^{\mathcal{J}} \subseteq P_2^{\mathcal{J}}$, and $P^{\mathcal{I}} = (P_1^- \circ P_2)^{\mathcal{J}}$, we can conclude that $(o, o'') \in P^{\mathcal{I}}$ and therefore, that $o \in (\exists P.B)^{\mathcal{I}}$.

($\Rightarrow$) Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a full model of $\mathcal{T}$, and let $role(\mathcal{T})$ denote the set of atomic roles in $\mathcal{T}$.

We show that $\mathcal{I}$ can be extended to a legal instantiation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ of $\Sigma(\mathcal{T})$, by assigning suitable extensions for the auxiliary classes and associations introduced in the construction of $\Sigma(\mathcal{T})$.

We set $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}} \cup \Gamma \cup \Lambda$, where

$$\Lambda = \biguplus_{A \sqsubseteq \forall P.B \in \mathcal{T}} \{a_{A_+}, a_{\bar{A}}\}\ , \text{ such that } \Delta^{\mathcal{I}} \cap \Lambda = \emptyset\,,$$

$$\Gamma = \biguplus_{P \in role(\mathcal{T})(\mathcal{T})} \Delta_P\ , \text{ with:}$$

$$\Delta_P = \{(o, o') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (o, o') \in P^{\mathcal{I}}\} \cup \bigcup_{A \sqsubseteq \forall P.B \in \mathcal{T}} \{(a_{A_+}, b), (a_{\bar{A}}, \bar{o})\},$$

with b an arbitrary instance of B, and $\bar{o}$ an arbitrary element of $\Delta^{\mathcal{I}}$.

We need to add for each assertion of the form $A \sqsubseteq \forall P.\, B$ a pair of fresh elements to $\Delta^{\mathcal{J}}$ in order to ensure that the extensions of all classes are nonempty. For instance, the extension of the class C_{PA+} might be empty if the elements in the extension of A are constrained by $\mathcal{T}$ in such a way that they do not have any P-successor. For this reason, we need to carefully define the extensions for the classes and associations used to encode the roles in $\mathcal{T}$.

We set $O^{\mathcal{J}} = \Delta^{\mathcal{I}} \cup \Lambda$, and $A^{\mathcal{J}} = A^{\mathcal{I}}$ for each class A representing an atomic concept, and $C_p^{\mathcal{J}} = \Delta_p$ for each $P \in role(\mathcal{T})$. Additionally, the extensions for the classes P_1 and P_2 are defined as follows:

$$P_1^{\mathcal{J}} = \{((o, o'), o) \mid (o, o') \in C_P^{\mathcal{J}}\}\ ,$$

$$P_2^{\mathcal{J}} = \{((o, o'), o') \mid (o, o') \in C_P^{\mathcal{J}}\}\ .$$

We now show that $\mathcal{J}$ is a full model of $\Sigma(\mathcal{T})$. Again, we analyze every case.

1. For each generalization between classes A and B in $\Sigma(\mathcal{T})$ introduced by an assertion of the form $A \sqsubseteq B$ in $\mathcal{T}$, $\mathcal{J}$ assigns extension to A and B, in such a way that $A^{\mathcal{J}} \subseteq B^{\mathcal{J}}$.

2. For each assertion of the form $A \sqsubseteq \neg B$ in $\mathcal{T}$. We have a fragment of $\Sigma(\mathcal{T})$ as in Figure 5.1a. $\mathcal{I}$ assigns to concepts A and B the subsets $A^{\mathcal{I}}$ and $B^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, such that, $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$. Then, we have that

   $$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \subseteq O^{\mathcal{J}}, \quad B^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \subseteq O^{\mathcal{J}}, \quad \text{and} \quad A^{\mathcal{J}} \cap B^{\mathcal{J}} = \emptyset$$

   which captures correctly the fragment of $\Sigma(\mathcal{T})$.

3. For each assertion of the form $A \sqsubseteq B_1 \sqcup B_2$, we have a fragment of $\Sigma(\mathcal{T})$ as in Figure 5.1b. $\mathcal{I}$ assigns to concepts B_1 and B_2 the subsets $B_1^{\mathcal{I}}$ and $B_2^{\mathcal{J}}$ of $\Delta^{\mathcal{I}}$, respectively, and to A a subset of their union $A^{\mathcal{I}} \subseteq B_1^{\mathcal{I}} \cup B_2^{\mathcal{I}}$.

Now, let us define $B^{\mathcal{J}} = B_1^{\mathcal{I}} \cup B_2^{\mathcal{I}}$. Clearly, $\mathcal{J}$ captures correctly the fragment of $\Sigma(\mathcal{T})$, as we have that

$$A^{\mathcal{J}} \subseteq B^{\mathcal{J}} ,$$

$$B_1^{\mathcal{J}} \subseteq B^{\mathcal{J}} ,$$

$$B_2^{\mathcal{J}} \subseteq B^{\mathcal{J}} ,$$

$$B^{\mathcal{J}} = B_1^{\mathcal{I}} \cup B_2^{\mathcal{I}} .$$

Furthermore, $B^{\mathcal{J}} \neq \emptyset$, as $B_1^{\mathcal{I}} \neq \emptyset$ and $B_2^{\mathcal{I}} \neq \emptyset$.

4. For the each assertion of the form $A \sqsubseteq \forall P.B$, let us define the extension for the *auxiliary* classes and associations as follows:

$$\begin{aligned}
A_+^{\mathcal{J}} &= A^{\mathcal{I}} \cup \{a_{A_+}\}, \\
\bar{A}^{\mathcal{J}} &= O^{\mathcal{J}} \setminus A_+^{\mathcal{J}}, \\
C_{PA_+}^{\mathcal{J}} &= \{(o, o') \in C_P^{\mathcal{J}} \mid o \in A_+^{\mathcal{J}}\}, \\
C_{P\bar{A}}^{\mathcal{J}} &= \{(o, o') \in C_P^{\mathcal{J}} \mid o \in \bar{A}^{\mathcal{J}}\}\}, \\
P_{A_{+1}}^{\mathcal{J}} &= \{((o, o'), o) \in P_1^{\mathcal{J}} \mid o \in A_+^{\mathcal{J}}\}, \\
P_{\bar{A}}^{\mathcal{J}} &= \{((o, o'), o) \in P_1^{\mathcal{J}} \mid o \in \bar{A}^{\mathcal{J}}\}, \\
P_{A_{+2}}^{\mathcal{J}} &= \{((o, o'), o) \in P_2^{\mathcal{J}} \mid o \in A_+^{\mathcal{J}}\}.
\end{aligned}$$

It is not difficult to see that $\mathcal{J}$ is a legal instantiation of a fragment of $\Sigma(\mathcal{T})$ as the one in Figure 5.2.

Now, it remains to show that each class and each association have a non-empty extension. This is clear for classes that encode atomic concepts in $\mathcal{T}$. For the classes A_+, $\bar{A}$, C_{PA_+}, and $C_{P\bar{A}}$ we have that

$$\begin{aligned}
a_{A_+} &\in A_+^{\mathcal{J}} , \\
a_{\bar{A}} &\in \bar{A}^{\mathcal{J}} , \\
(a_{A_+}, b) &\in C_{PA_+}^{\mathcal{J}} , \\
(a_{\bar{A}}, o) &\in C_{P\bar{A}}^{\mathcal{J}} .
\end{aligned}$$

For the associations $P_1, P_2, P_{A_{+1}}, P_{A_{+2}}$ and $P_{\bar{A}}$ we have that

$$\begin{aligned}
((a_{A_+}, b), a_{A_+}) &\in P_{A_{+1}}^{\mathcal{J}} \subseteq P_1^{\mathcal{J}} , \\
((a_{A_+}, b), b) &\in P_{A_{+2}}^{\mathcal{J}} \subseteq P_2^{\mathcal{J}} , \\
((a_{\bar{A}}, o), a_{\bar{A}}) &\in P_{\bar{A}}^{\mathcal{J}} .
\end{aligned}$$

5. For each assertion of the form $A \sqsubseteq \exists P.B$, let us to define the extensions for the auxiliary classes and associations as follows:

$$\begin{aligned} C^{\mathcal{J}}_{P_{AB}} &= \{(o,o') \in \Delta_P \mid o \in A^{\mathcal{I}} \text{ and } o' \in B^{\mathcal{I}}\}\,, \\ P^{\mathcal{J}}_{AB1} &= \{((o,o'),o) \in P_1^{\mathcal{J}} \mid (o,o') \in C^{\mathcal{J}}_{P_{AB}}\}\,, \\ P^{\mathcal{J}}_{AB2} &= \{((o,o'),o') \in P_2^{\mathcal{J}} \mid (o,o') \in C^{\mathcal{J}}_{P_{AB}}\}\,. \end{aligned}$$

We have that $C^{\mathcal{J}}_{P_{AB}} \neq \emptyset$ as there exists a pair $(a,b) \in \Delta_P$ with $a \in A^{\mathcal{I}}$, and $b \in B^{\mathcal{I}}$. Since $C^{\mathcal{J}}_{P_{AB}} \neq \emptyset$, we have that $P^{\mathcal{J}}_{AB1} \neq \emptyset$ and $P^{\mathcal{J}}_{AB2} \neq \emptyset$. □

Theorem 5.6. *Full satisfiability of* UML_{isaR} *CDs is* ExpTime*-complete*

Proof:

- The upper complexity bound can be established by reducing full consistency of an UML_{isaR} CD to class consistency of UML CD, which is known to be ExpTime-complete (Berardi et al., 2005).

  Given an UML_{isaR} CD $\mathcal{D}$, with classes $C_1, \ldots, C_n$, we construct the UML CD $\mathcal{D}'$ by adding to $\mathcal{D}$:

  – a new class C_{top},
  – a new association R_i for $i \in \{1, \ldots, n\}$,

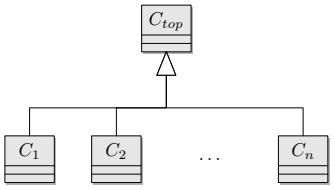
  Besides, in order to ensure that every association is also populated, we consider each association P between the classes C_i and C_j such that neither C_i nor C_j is constrained to participate at least once in P. We add two new associations, R_P and P' and a new class C_i^P. Finally, we add the constraints shown in Figure 5.4. Clearly, we have that $\mathcal{D}$ is fully satisfiable if and only if the class C_{top} is satisfiable in $\mathcal{D}'$.

- The lower complexity bound follows from Lemma 5.5 □

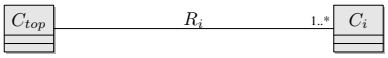
5.2.2 Full satisfiability in UML_{Bool}

In this section, we investigate the complexity of the full satisfiability problem for UML_{Bool} (defined in Section 4.3. The techniques are similar to the satisfiability proofs presented in (Artale et al., 2007a).

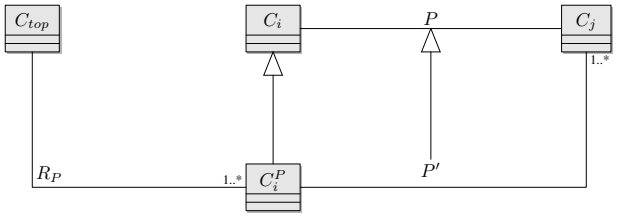
We show that deciding full satisfiability of UML_{Bool} diagrams is NP-complete. We provide a polynomial reduction of the 3Sat problem (which is known to be NP-complete, Papadimitriou 1994) to full satisfiability of UML_{Bool} CDs.



(a) Every class inherits from C_{top}



(b) Every instance of C_{top} is related with at least one instance of C_i, for $i \in \{1, \ldots, n\}$



(c)

Figure 5.4: Reducing UML full consistency to class consistency

Let an instance of 3SAT be given by a set $\phi = \{c_1, \ldots, c_m\}$ of 3-clauses over a finite set Π of propositional variables, such that $c_i = l_i^1 \vee l_i^2 \vee l_i^3$, for $i \in \{1, \ldots, m\}$, where each l_j^k is a literal, i.e., a variable or its negation. We construct an UML$_{Bool}$ diagram $\mathcal{D}$ as follows:

- $\mathcal{D}$ contains the classes C_p and $C_{\neg p}$ for each variable $p \in \Pi$,
- $\mathcal{D}$ contains one class C_i for each clause $c_i \in \phi$,
- $\mathcal{D}$ contains the classes C_ϕ and $C_\top$

The constraints imposed by $\mathcal{D}$ are given by the following TBox $\mathcal{T}_\mathcal{D}$. For every $i \in \{1, \ldots, m\}$, $j \in \{1, 2, 3\}$ and $p \in \Pi$, we have the assertions

$$C_\phi \sqsubseteq C_i \tag{5.1}$$

$$C_i \sqsubseteq C_\top \tag{5.2}$$

$$C_l \sqsubseteq C_\top \tag{5.3}$$

$$C_{l_i^j} \sqsubseteq C_i \tag{5.4}$$

$$C_{\neg p} \sqsubseteq \neg C_p \tag{5.5}$$

$$C_\top \sqsubseteq C_p \sqcup C_{\neg p} \tag{5.6}$$

$$C_i \sqsubseteq C_{l_i^1} \sqcup C_{l_i^2} \sqcup C_{l_i^3} \tag{5.7}$$

Lemma 5.7. *A set ϕ of 3-clauses is satisfiable if and only if the* UML_{Bool} *CD $\mathcal{D}_\phi$, constructed as above, is fully satisfiable.*

Proof: ($\Rightarrow$) Let $\mathcal{J} \models \phi$. Define an interpretation $\mathcal{I} = (\{0,1\}, \cdot^\mathcal{I})$, with

$$C_\top^\mathcal{I} = \{0,1\}$$

$$C_l^\mathcal{I} = \begin{cases} \{1\} & \text{if } \mathcal{J} \models l \\ \{0\} & \text{otherwise} \end{cases}$$

$$C_i^\mathcal{I} = \bigcup_{k\in\{1,2,3\}} C_{l_i^k}^\mathcal{I} \quad \text{for } c_i = l_i^1 \vee l_i^2 \vee l_i^3$$

$$C_\phi^\mathcal{I} = \bigcap_{1\le i\le m} C_i.$$

Clearly, $C^\mathcal{I} \neq \emptyset$ for every class C representing a clause or a literal, and for $C = C_\top$. Moreover, as at least one literal l_i^j in each clause is such that $\mathcal{J} \models l_i^j$, then $1 \in C_i^\mathcal{I}$ for every $i \in 1, \ldots, m$, and therefore $1 \in C_\phi^\mathcal{I}$. It is straightforward to check that $\mathcal{I}$ satisfies $\mathcal{T}$.

($\Leftarrow$) Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be a fully satisfiable model of $\mathcal{D}_\phi$. Construct a model $\mathcal{J}$ of ϕ by taking an element $o \in C_\phi^\mathcal{I}$, and setting, for every variable $p \in \Pi$, $\mathcal{J} \models p$ iff $o \in C_p^\mathcal{I}$. Let us show that $\mathcal{J} \models \phi$. Indeed, as $o \in C_\phi^\mathcal{I}$ and by the generalization in (5.1), we have that $o \in C_i^\mathcal{I}$ for $1 \le i \le m$. By the completeness constraint in (5.6), there is some $j \in \{1,2,3\}$ such that $o \in C_{l_i^j}$ for every $i \in \{1, \ldots, m\}$. If l_i^j is a variable, then $\mathcal{J} \models l_i^j$ by construction,

and thus $\mathcal{J} \models c_i$. Otherwise, if $l_i^k = \neg p$ for some variable p, then, by (5.5), $o \notin C_p^{\mathcal{I}}$. Thus, $\mathcal{J} \models \neg p$, and therefore, $\mathcal{J} \models c_i$. □

Theorem 5.8. *Full satisfiability of* UML_{Bool} *is* NP*-complete*

Proof: Using a result in (Artale et al., 2007a), it can be shown that class/association satisfiability for UML_{Bool} CDs is in NP. Every UML_{Bool} CDs can be encoded into a *DL-Lite*$_{Bool}$ (Artale et al., 2007b) TBox, in which every class/association is encoded by an atomic concept. The encoding is such that a class (association) C is satisfiable if and only if the corresponding concept C is satisfiable w.r.t. the TBox encoding the diagram. The NP membership follows then from the NP membership of concept satisfiability w.r.t. a *DL-Lite*$_{Bool}$ TBox (Artale et al., 2007b).

Finally, full satisfiability of UML_{Bool} diagrams can be reduced to checking satisfiability of every class and association in the diagram. If for every class C, there is a model of the diagram in which C is satisfied, then the (disjoint) union of all such models is a full model of the diagram. The argument holds due to the special character of UML CDs constraints, which are invariant under disjoint model union (Lenzerini and Nobili, 1990; Maraee and Balaban, 2006).

The NP-hardness follows from Lemma 5.7. □

5.2.3 Full satisfiability in UML_{ref}

We turn now to UML_{ref} CDs, and show that full satisfiability is NLogSpace-complete. We provide a reduction of the REACHABILITY problem on (acyclic) directed graphs, which is known to be NLogSpace-complete (see e.g., Papadimitriou, 1994) to (the complement of) full-satisfiability of UML_{ref} CDs.

Let $G = (V, E, s, t)$ be an instance of REACHABILITY, where V is a set of vertices, $E \subseteq V \times V$ is a set of directed edges, and s, t are the initial and terminal vertices respectively. Construct an UML_{ref} CD $\mathcal{D}_G$ from G as follows:

- $\mathcal{D}_G$ has two classes C_v^1 and C_v^2, for each vertex $v \in V \setminus \{s\}$, and one class C_s corresponding to the initial vertex s,
- for each edge $(u, v) \in E$ with $u \neq s$ and $v \neq s$, $\mathcal{D}_G$ has the following constraints:

$$\begin{aligned} C_u^1 &\sqsubseteq C_v^1 \\ C_u^2 &\sqsubseteq C_v^2 \end{aligned}$$

- for each edge $(s, v) \in E$, $\mathcal{D}_G$ contains the following generalization constraints:

$$\begin{aligned} C_s &\sqsubseteq C_v^1 \\ C_s &\sqsubseteq C_v^2 \end{aligned}$$

- for each edge $(u, s) \in E$, $\mathcal{D}_G$ imposes the generalization constraints:

$$\begin{aligned} C_u^1 &\sqsubseteq C_s \\ C_u^2 &\sqsubseteq C_s \end{aligned}$$

- the classes C_t^1 and C_t^2 are constrained to be disjoint in $\mathcal{D}$, expressed by:

$$C_t^1 \sqsubseteq \neg C_t^2$$

We show now the following Lemma.

Lemma 5.9. *t is reachable from s in G iff $\mathcal{D}_G$ is not fully satisfiable.*

Proof: ($\Rightarrow$) Let $\pi = v_1, \ldots, v_n$ be a path in G with $v_1 = s$ and $v_n = t$. We claim that the class C_s in the constructed diagram $\mathcal{D}_G$ is unsatisfiable. Suppose otherwise, that there is a model $\mathcal{I}$ of $\mathcal{D}_G$ with $C_s^{\mathcal{I}} \neq \emptyset$. From π, the construction yields a number of generalization constraints in $\mathcal{D}_G$ such that the following holds:

$$C_s^{\mathcal{I}} \subseteq \cdots \subseteq (C_t^1)^{\mathcal{I}} \tag{5.8}$$
$$C_s^{\mathcal{I}} \subseteq \cdots \subseteq (C_t^2)^{\mathcal{I}} \tag{5.9}$$

From (5.8) and (5.9), we obtain that $d \in (C_t^1)^{\mathcal{I}}$ and $d \in (C_t^2)^{\mathcal{I}}$, which violates the disjointness between the classes C_t^1 and C_t^2, in contradiction to $\mathcal{I}$ being a model of $\mathcal{D}_G$. Hence, C_s is unsatisfiable, and therefore $\mathcal{D}_G$ is not fully satisfiable.

($\Leftarrow$) Let us consider the contrapositive. Assume that t is not reachable from s in G. We construct a full model $\mathcal{I}$ of $\mathcal{D}_G$. Let $\Delta^{\mathcal{I}} = \{d_s\} \cup \bigcup_{v \in V \setminus \{s\}} \{d_v^1, d_v^2\}$. Define inductively a sequence of interpretations as follows:

$\mathcal{I}^0 := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}^0})$, such that:

$$\begin{aligned} (C_s)^{\mathcal{I}^0} &:= \{d_s\}, \\ (C_v^i)^{\mathcal{I}^0} &:= \{d_v^i\}, \text{ for all } i \in \{1, 2\}, v \in V \setminus \{s\}\,. \end{aligned}$$

$\mathcal{I}^{n+1} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}^{n+1}})$, such that

$$(C_s)^{\mathcal{I}^{n+1}} := C_s^{\mathcal{I}^n} \cup \bigcup_{(u,s)\in E} ((C_u^1)^{\mathcal{I}^n} \cup (C_u^2)^{\mathcal{I}^n})$$

$$(C_v^i)^{\mathcal{I}^{n+1}} := (C_v^i)^{\mathcal{I}^n} \cup \bigcup_{(u,v)\in E, u\neq s} (C_u^i)^{\mathcal{I}^n} \cup \bigcup_{(s,v)\in E} (C_s)^{\mathcal{I}^n}$$

The definition induces a monotone operator over a complete lattice, and hence it has a fixed point. Let $\mathcal{I}$ be defined by such a fixed point. It is easy to check that $\mathcal{I}$ is such that for all $i \in \{1, 2\}$, and $u, v \in V \setminus \{s\}$ the following holds:

1. For each class C_v^i, we have that $d_v^i \in (C_v^i)^{\mathcal{I}}$,
2. $d_s \in C_s^{\mathcal{I}}$,
3. For all $d \in \Delta^{\mathcal{I}}$, $d \in (C_u^i)^{\mathcal{I}}$ implies $d \in (C_v^i)^{\mathcal{I}}$ iff v is reachable from u in G,
4. For all $d_u^i \in \Delta^{\mathcal{I}}$, $d_u^i \in (C_v^j)^{\mathcal{I}}$ for $i \neq j$ iff s is reachable from u in G, and v is reachable from s in G,
5. $d_s \in C_v^i$ iff v is reachable from s in G.

From (1) and (2) we have that all classes in $\mathcal{D}_G$ are populated in $\mathcal{I}$. It remains to show that $\mathcal{I}$ satisfies $\mathcal{D}_G$. A generalization between the classes C_u^i and C_v^i corresponds to the edge $(u, v) \in E$. This means that v is reachable from u in G, and therefore, we have by (3) that $(C_u^i)^{\mathcal{I}} \subseteq (C_v^i)^{\mathcal{I}}$. A similar argument holds for generalizations involving the class C_s. Furthermore, the classes C_t^1 and C_t^2 are disjoint under $\mathcal{I}$. Suppose that there is an element $d \in \Delta^{\mathcal{I}}$ such that $d \in (C_t^1)^{\mathcal{I}} \cap (C_t^2)^{\mathcal{I}}$. Then by (5) $d \neq d_s$, as t is not reachable from s. Moreover, $d \neq d_v^i$ for some $i \in \{1, 2\}$ and $v \in V \setminus \{s\}$. Indeed, suppose w.l.o.g. that $i = 1$. Then, by (4), $d_v^1 \in (C_t^2)^{\mathcal{I}}$ iff s is reachable from v and t is reachable from s, which leads to a contradiction. Hence, $C_t^1 \cap C_t^2 = \emptyset$. □

Theorem 5.10. *Full-satisfiability of* UML_{ref} *CDs is* NLogSpace*-complete.*

Proof: The NLogSpace membership follows from the NLogSpace membership of class satisfiability (Artale et al., 2007a), and a similar argument as for the upper bound in Theorem 5.8. As NLogSpace = coNLogSpace (by the Immerman-Szelepcsényi theorem; (see e.g., Papadimitriou, 1994), and as the above reduction is logspace bounded, it follows that full satisfiability of UML_{ref} CDs is NLogSpace-hard. □

Chapter 6

Conclusions

The objective of this thesis was to address the complexity of full satisfiability of UML class diagrams. To this end, we studied the existing result of class satisfiability and the methods implemented for checking full satisfiability (c.f. Chapter 4). The results in Chapter 5, confirm the intuition that the complexity of full satisfiability of UML class diagrams coincides with that of class satisfiability. We have provided upper and lower complexity bounds for UML class diagrams with diverse expressiveness:

UML_{isaR}	Class diagrams that include completenesses and disjointness constraints over class generalizations, but restrict to unconstrained generalizations of associations.
UML_{Bool}	Class diagrams that restrict to generalizations over classes (no association classes), but allow for completeness and disjointness constraints.
UML_{ref}	Class diagrams that restrict to generalizations over classes and allow only for disjointness constraints. Additionally, it is possible to express refinement of multiplicity constraints of subclasses participating on associations.

In each case, the upper complexity bounds are almost a direct consequence of the corresponding upper bounds of class satisfiability derived for the encoding of UML CDs into description logics TBoxes, and the disjoint union model property of description logics.

On the other hand, the lower bounds required a careful analysis and adaptation of the corresponding proofs for class satisfiability. In particular:

- We defined a notion for full satisfiability of $\mathcal{ALC}$ TBoxes, and proved that deciding full satisfiability of these TBoxes is ExpTime-complete, i.e., these problems coincides in complexity with the standard reasoning in $\mathcal{ALC}$.

- We adapted the encoding of $\mathcal{ALC}$ into UML class diagrams in (Artale et al., 2007a; Berardi et al., 2005), for proving the ExpTime-hardness of full satisfiability of UML_{isaR} class diagrams. We then reduced full satisfiability of $\mathcal{ALC}$ KBs to full satisfiability of UML_{isaR} class diagrams.
- We reduced the 3-SAT problem to full satisfiability of UML_{Bool} class diagrams, and prove then that full satisfiability is NP-hard.
- We reduced the REACHABILITY problem on acyclic directed graphs, to full satisfiability of UML_{ref} class diagrams, and prove then that this problem is NLogSpace-hard.

Possible extension for the results on this thesis, include the study of the complexity of finite full satisfiability. In a more practical approach, the development of methods for determining not only full satisfiability but also for computing the set of unsatisfiable classes and associations would be of interest.

Bibliography

Pascal André, Annya Romanczuk, and Jean-Claude Royer. Checking the Consistency of UML Class Diagrams Using Larch Prover. In *Rigorous Object-Oriented Methods*, Workshops in Computing. BCS, 2000a.

Pascal André, Annya Romanczuk, Jean-Claude Royer, and Aline Vasconcelos. An algebraic view of UML class diagrams. In Christophe Dony and Houari A. Sahraoui, editors, *LMO*, pages 261–276. Hermès, 2000b. ISBN 2-7462-0093-7.

Alessandro Artale, Francesca Cesarini, and Giovanni Soda. Describing Database Objects in a Concept Language Environment. *IEEE Trans. Knowl. Data Eng.*, 8(2):345–351, 1996.

Alessandro Artale, Diego Calvanese, Roman Kontchakov, Vladislav Ryzhikov, and Michael Zakharyaschev. Reasoning over Extended ER Models. In *Proc. of the 26th Int. Conf. on Conceptual Modeling (ER 2007)*, volume 4801 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2007a.

Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. DL-Lite in the Light of First-Order Logic. In *Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI 2007)*, pages 361–366, 2007b.

Franz Baader and Ulrike Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69(1):5–40, 2001.

Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

Mira Balaban and Azzam Maraee. Consistency of UML Class Diagrams with Hierarchy Constraints. In Opher Etzion, Tsvi Kuflik, and Amihai Motro, editors, *NGITS*, volume 4032 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 2006. ISBN 3-540-35472-7.

Mira Balaban and Azzam Maraee. Efficient Reasoning About Finite Satisfiability of UML Class Diagrams with Constrained Generalization Sets. In David H. Akehurst, Régis Vogel, and Richard F. Paige, editors, *ECMDA-FA*, volume 4530 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2007. ISBN 978-3-540-72900-6.

Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML Class Diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005.

Grady Booch. *Object-Oriented Design with Applications*. Benjamin/Cummings, 1990.

Alex Borgida and Ronald J. Brachman. Conceptual Modeling with Description Logics. In Baader et al. (2003), pages 349–372. ISBN 0-521-78176-0.

Alexander Borgida, Maurizio Lenzerini, and Riccardo Rosati. Description logics for databases. In Baader et al. (2003), pages 462–484.

Marco Cadoli, Diego Calvanese, Giuseppe De Giacomo, and Toni Mancini. Finite Satisfiability of UML Class Diagrams by Constraint Programming. In *In Proc. of the CP 2004 Workshop on CSP Techniques with Immediate Application*, 2004.

Diego Calvanese and Giuseppe De Giacomo. Expressive Description Logics. In Baader et al. (2003), pages 178–218.

Diego Calvanese and Maurizio Lenzerini. On the Interaction Between ISA and Cardinality Constraints. In *ICDE*, pages 204–213. IEEE Computer Society, 1994. ISBN 0-8186-5400-7.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description Logic Framework for Information Integration. In *KR*, pages 2–13, 1998a.

Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description Logics for Conceptual Data Modeling. In Jan Chomicki and Gunter Saake, editors, *Logics for Databases and Information Systems*, pages 229–263. Kluwer, 1998b.

Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Unifying Class-Based Representation Formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.

Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification Constraints and Functional Dependencies in Description Logics. In Bernhard Nebel, editor, *IJCAI*, pages 155–160. Morgan Kaufmann, 2001a. ISBN 1-55860-777-3.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi. Reasoning in Expressive Description Logics. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1581–1634. Elsevier and MIT Press, 2001b. ISBN 0-444-50813-9, 0-262-18223-8.

Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable Description Logics for Ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in Description Logics. In *Principles of Knowledge Representation*, pages 191–236. Center for the Study of Language and Information, Stanford, CA, USA, 1996. ISBN 1-57586-056-2.

Enrico Franconi and Gary Ng. The i•com tool for Intelligent Conceptual Modeling. In Mokrane Bouzeghoub, Matthias Klusch, Werner Nutt, and Ulrike Sattler, editors, *KRDB*, volume 29 of *CEUR Workshop Proceedings*, pages 45–53. CEUR-WS.org, 2000.

Martin Gogolla and Mark Richters. Expressing UML Class Diagrams Properties with OCL. In Tony Clark and Jos Warmer, editors, *Object Modeling with the OCL*, volume 2263 of *Lecture Notes in Computer Science*, pages 85–114. Springer, 2002. ISBN 3-540-43169-1.

Volker Haarslev and Ralf Möller. RACER System Description. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *IJCAR*, volume 2083 of *Lecture Notes in Computer Science*, pages 701–706. Springer, 2001. ISBN 3-540-42254-4.

Sven Hartmann. Graph-Theoretical Methods to Construct Entity-Relationship Databases. In Manfred Nagl, editor, *WG*, volume 1017 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 1995. ISBN 3-540-60618-1.

Xudong He. Formalizing UML Class Diagrams: A Hierarchical Predicate Transition Net Approach. In *COMPSAC*, pages 217–222. IEEE Computer Society, 2000. ISBN 0-7695-0792-1.

Ian Horrocks. Implementation and Optimization Techniques. In Baader et al. (2003), pages 306–346.

Ian Horrocks. FaCT. In Enrico Franconi, Giuseppe De Giacomo, Robert M. MacGregor, Werner Nutt, and Christopher A. Welty, editors, *Description Logics*, volume 11 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1998.

Ian Horrocks. FaCT and iFaCT. In Patrick Lambrix, Alexander Borgida, Maurizio Lenzerini, Ralf Möller, and Peter F. Patel-Schneider, editors, *Description Logics*, volume 22 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1999.

Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical Reasoning for Expressive Description Logics. In Harald Ganzinger, David A. McAllester, and Andrei Voronkov, editors, *LPAR*, volume 1705 of *Lecture Notes in Computer Science*, pages 161–180. Springer, 1999. ISBN 3-540-66492-0.

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

Soon-Kyeong Kim and David A. Carrington. Formalizing the UML Class Diagram Using Object-Z. In Robert B. France and Bernhard Rumpe, editors, *UML*, volume 1723 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 1999.

Maurizio Lenzerini and Paolo Nobili. On the satisfiability of dependency constraints in Entity-Relationship schemata. *Inf. Syst.*, 15(4):453–461, 1990.

A. Maraee and M. Balaban. Efficient Decision of Consistency in UML Diagrams with Constrained Generalization Sets. In *1st Workshop on Quality in Modeling, MoDELS*, 2006.

Azzam Maraee. Efficient Methods for solving finite satisfiability problems in UML class diagrams. Master's thesis, Ben-Gurion University of the Negev, 2007.

Marvin Minsky. A Framework for Representing Knowledge. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.

Ralf Möller and Volker Haarslev. Description Logic Systems. In Baader et al. (2003), pages 282–305.

Bernhard Nebel. Terminological Reasoning is Inherently Intractable. *Artif. Intell.*, 43(2):235–249, 1990.

Christos M. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, 1994. ISBN 0201530821.

Anna Queralt and Ernest Teniente. Reasoning on UML Class Diagrams with OCL Constraints. In David W. Embley, Antoni Olivé, and Sudha Ram, editors, *ER*, volume 4215 of *Lecture Notes in Computer Science*, pages 497–512. Springer, 2006. ISBN 3-540-47224-X.

M. Quillian. Semantic Memory. In M. Minsky, editor, *Semantic Information Processing*, pages 227–270. MIT Press, 1968.

Mark Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universität Bremen, Logos Verlag, Berlin, BISS Monographs, No. 14, 2002.

James E. Rumbaugh, Michael R. Blaha, William J. Premerlani, Frederick Eddy, and William E. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991. ISBN 0-13-630054-5.

Ken Satoh, Ken Kaneiwa, and Takeaki Uno. Contradiction Finding and Minimal Recovery for UML Class Diagrams. In *ASE*, pages 277–280. IEEE Computer Society, 2006. ISBN 0-7695-2579-2.

Manfred Schmidt-Schauß and Gert Smolka. Attributive Concept Descriptions with Complements. *Artif. Intell.*, 48(1):1–26, 1991.

Malcolm Shroff and Robert B. France. Towards a formalization of UML class structures in Z. In *COMPSAC*, pages 646–. IEEE Computer Society, 1997. ISBN 0-8186-8105-5.

Marcin Szlenk. Formal Semantics and Reasoning about UML Class Diagram. In *DepCoS-RELCOMEX*, pages 51–59. IEEE Computer Society, 2006. ISBN 0-7695-2565-2.

Bernhard Thalheim. Fundamentals of Cardinality Constraints. In Günther Pernul and A. Min Tjoa, editors, *ER*, volume 645 of *Lecture Notes in Computer Science*, pages 7–23. Springer, 1992. ISBN 3-540-56023-8.

Bernhard Thalheim. Foundations of Entity - Relationship Modeling. *Ann. Math. Artif. Intell.*, 7(1-4):197–256, 1993.

UML. *Unified Modeling Language Specification*. Framingham, Mass., 1998. URL www.omg.org.

Jos B. Warmer and Anneke G. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, Reading Massachusetts, 1999.

Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener. *Designing Object- Oriented Software*. Prentice Hall, Englewood Cliffs, N.J, 1990.