Università degli Studi di Roma "La Sapienza"
Dottorato di Ricerca in Informatica

Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms

Diego Calvanese

VIII-96-2

Diego Calvanese

Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms

Dottorato di Ricerca in Informatica

VIII-96-2

Università degli Studi di Roma "La Sapienza"

Author's address:
Diego Calvanese
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
e-mail: calvanese@dis.uniroma1.it
www: http://www.dis.uniroma1.it/~calvanes

Contents

Acknowledgments

I wish to acknowledge the people who helped me most during my Ph.D..

First of all, my deepest debt of gratitude goes to Maurizio Lenzerini, who has been a great advisor who constantly inspired and encouraged me. It is mainly due to him if I could accomplish and finish this work.

I wish to thank the colleagues with whom I worked in the past years, and in particular my coauthors Maurizio Lenzerini, Giuseppe De Giacomo, and Daniele Nardi. A special thank to Giuseppe for many interesting discussions that contributed directly to the results in this thesis.

I would like to thank all the people of the Dipartimento di Informatica e Sistemistica of the University of Rome "La Sapienza", and in particular the members of the Artificial Intelligence group. I shared with them joys and pains of a working environment which has always been stimulating and which sometimes surprised us with unexpected events. A special thank goes to Andrea Schaerf with whom I shared for some time even desk and workstation, and to Fiora Pirri for her constant support.

I am especially indebted to Franz Baader, who gave me his hospitality for nine months at the Lehr- und Forschungsgebiet Theoretische Informatik of the RWTH Aachen, and who introduced me to the fascinating topic of automata on infinite objects. I thank my colleagues in Aachen, Ulrike Sattler, Can Adam Albayrak, Jöern Richts and also Sonja Erli Hagemayer, who arranged everything for my stay in Aachen. They all offered me much more than just a friendly and pleasant working environment.

I express my gratitude to the members of the Ph.D. Committee of the Dipartimento di Informatica e Sistemistica, who helped me in the accomplishment of this work. To Alexander Borgida and Jan Van den Bussche as external referees, to Paolo Atzeni and Eugenio Omodeo as internal referees, and to Luigia Carlucci Aiello as president. Especially Eugenio and Alexander gave me many valuable comments on an earlier draft of the thesis.

I also would like to thank ESPRIT Basic Research Action N.6810 (COMPULOG 2) and Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of the CNR (Italian Research Council), who financed my journeys around the world and provided me the working equipment.

Last but not least, special thanks to my loving wife Paola, who encouraged me and gave me strength in difficult moments.

April 30, 1996 Diego Calvanese

Abstract

The idea of developing representation systems based on a structured representation of knowledge was first pursued in Knowledge Representation with Semantic Networks and Frames. Later, *Description Logics* have been introduced with the aim of providing a simple and well-established first-order semantics to capture the meaning of the most popular features of the structured representations of knowledge. This research has paralleled the one carried out in Databases, where first semantic and conceptual database models and more recently object-oriented database models have been developed, which allow for an abstract and modular representation of data relationships at the intensional level, overcoming some limitations of record-based data models. However, while in Description Logics the emphasis has always been on reasoning on and about structured descriptions, this has not been really the issue in database models, where emphasis was more on accurately capturing the complex relations that occur between data in typical commercial applications.

In this thesis we try to integrate these two lines of research, by devising in an incremental manner a set of abstract representation languages, called $\mathcal{L}$-*languages*, and studying reasoning on them. Using $\mathcal{L}$-languages one can build (by means of certain constructors that characterize the language) complex class and attribute expressions and use these to define schemata for the representation of knowledge at the intensional level. The meaningful constructors we consider in this thesis are arbitrary boolean combinations of classes, cardinality restrictions, which generalize functional and existence constraints in Databases, inverse attributes and more complex traversal patterns, and finally well-foundedness constraints, which allow one to define several recursive and inductive structures.

We first show that $\mathcal{L}$-languages are indeed adequate for our purposes, since they subsume (in their more expressive variants) the structural aspects of most of the representation formalisms used both in Knowledge Representation and in Databases.

The main part of the thesis is concerned with the design of effective techniques for intensional reasoning on $\mathcal{L}$-schemata and the study of their computational properties. The reasoning tasks we consider are *schema consistency* ("Is there a nonempty database that satisfies all constraints expressed in the schema?"), *class consistency*, ("Can a class be populated in a database satisfying the schema?"), and *class subsumption* ("Does a class denote a subset of another class in every database satisfying the schema?").

We first characterize the intrinsic complexity of these reasoning tasks for a relevant class of languages and different types of schemata. In particular, even for the least expressive $\mathcal{L}$-language and the simplest form of schemata, which are subsumed by most known representation formalisms, reasoning is already intractable. Adding either the possibility to express disjunction or the possibility to use cycles in the schema, further increases the computational complexity. We show also that the combination of these two features is already sufficient to make reasoning **EXPTIME**-hard, i.e. as hard as reasoning on the most general type of schemata using the most expressive language.

The interaction that occurs between some of the constructs in the more expressive $\mathcal{L}$-languages when they are used inside schemata causes the finite model property not to hold. This means that reasoning differs in the case when we consider arbitrary models (i.e. databases of arbitrary, possibly infinite, size) from the case when we restrict our attention to finite ones. The latter assumption is in fact the common one in databases, while finiteness of the underlying domain has seldom been an issue in Knowledge Representation. For this reason we analyze reasoning in both contexts separately.

In the case of unrestricted models the reasoning techniques developed in Knowledge Representation are adequate. They are based on the similarity that exists between the interpretative structures of $\mathcal{L}$-languages and those of Propositional Dynamic Logics (PDLs), which are formalisms specifically designed for reasoning about program schemes. Such similarity results in a well-known correspondence between both formalisms which we extend in this work in order to cope with all the constructs present in $\mathcal{L}$-languages. We show

decidability in deterministic exponential time for an extended version of PDL which corresponds to the most expressive $\mathcal{L}$-language we consider. The results are obtained by extending known techniques based on automata on infinite trees, which exploit the existence of particular tree-like models for satisfiable formulae of the PDL. The correspondence allows us to establish the same (tight) upper bound for reasoning with respect to arbitrary models in schemata expressed in this language.

In the case of finite models, on the other hand, the known reasoning methods cannot be used, since the existence of tree-like models is not guaranteed if the domain has to be finite. For this reason completely new reasoning techniques are developed. They are based on the construction from a given schema of a system of linear inequalities, and on the relationship that exists between particular solutions of this system and finite models of the schema. The resulting algorithm works in worst case deterministic exponential time for a relevant subset of $\mathcal{L}$-schemata, and exhibits efficient behaviour in meaningful practical cases. A deterministic double exponential upper bound is established for the most expressive form of $\mathcal{L}$-schemata.

Chapter 1

Representing Structured Information

Current commercial applications require the manipulation of large amounts of data with complex interrelationships. Therefore, the study and development of formalisms that allow for a structured representation of knowledge is gaining an ever increasing importance.

Such representation formalisms have been developed separately in different fields, with different motivations and needs. On one side, in Databases, researchers have been looking for expressive data models with which they could represent data that is manipulated by commercial applications. The aim was to use these models in the analysis and design phase of complex information systems. This led to the development of conceptual and semantic data models, in which the emphasis is exactly on the accurate representation of complex data relationships. More recently, the need to create a tighter coupling between database management systems and modern programming languages based on the object-oriented paradigm, has led to the development of so called object-oriented data models.

Researchers working in Artificial Intelligence have recognized very early that any machine that should exhibit an "intelligent behaviour", needs to handle a great amount of information. One of the central problems to solve is finding mechanisms for representing knowledge in such a way that a mechanized agent can manipulate it and use it for performing deductions in an efficient way. Therefore Knowledge Representation has become a research field itself [112], and has led to the development of a great variety of formalisms for modeling knowledge in a structured way, and for manipulating it by using formal tools.

Research in Knowledge Representation is based on what Smith [148] calls the *Knowledge Representation Hypothesis*:

> "Any mechanically embodied intelligent process will be comprised of structural ingredients that (a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and (b) independent of such external semantical attribution, play a formal but causal and essential role in engendering the behaviour that manifests that knowledge."

Knowledge based systems can be considered as those systems that satisfy the Knowledge Representation Hypothesis by definition. This means that they contain a distinct component, called the Knowledge Representation (sub)system, which manipulates a knowledge base and exhibits to the outside a picture of the world that corresponds to what is represented in the knowledge base. From the birth of the discipline to now a great variety of formalisms for the definition and manipulation of knowledge bases have been studied and developed.

In the rest of the chapter we introduce the most important formalisms for the representation of structured knowledge used both in Artificial Intelligence and in Databases: In Section 1.1 we describe Semantic Networks and Frame based systems, and in Section 1.2 we discuss Description Logics. Semantic and object-oriented database models are briefly introduced in Sections 1.3 and 1.4, respectively. We conclude the chapter by describing the goal of the thesis and giving an outlook on the main results.

1.1 Semantic Networks and Frame Based Systems

Semantic Networks were one of the first attempts at structured knowledge representation. The concept of Semantic Network was introduced in [130] by Quillian, who intended to give a formal representation of the "objective" meaning of words by means of his *Semantic Memory Model*. Although the model he proposed was very simple, if compared to the successive developments, it introduced a number of concepts that have successively been recognized as fundamental for the whole field of Knowledge Representation.

A Semantic Network represents knowledge by means of a set of nodes that are connected to each other by links of different types. Each node corresponds either to a concept or to a real world object, and the connections between nodes represent the relationships that exist between concepts and/or objects.

As Woods in [167] and Brachman in [30] point out, despite their name Semantic Networks did not have a precise semantics. This is partly due to the fact that in a Semantic Network both arcs and nodes can carry different types of meaning: A node can represent either a concept, with an associated set of properties that hold for all real world objects that are an instance of it. Or it can represent a single object with specific properties. Similarly, a link between two nodes can either contribute to the definition of a concept, or it can represent a relation that holds between two objects and is therefore a specific assertion about the world. Nodes and links of the first type carry *intensional* knowledge, while those of the second type provide *extensional* knowledge. This distinction is in fact fundamental and at the base of all recent knowledge representation formalisms.

The lack of a clear semantics strongly limits the use of Semantic Networks as knowledge representation component for applications in Artificial Intelligence. As pointed out by Hayes in [91], a knowledge representation language can be characterized as a formal language which has a semantic theory, by which Hayes means an account of how the expressions of the language relate to the real world objects about which the language claims to express knowledge. This implies that a knowledge base, built out of expressions of a knowledge representation language, is given a semantics a priori and "carries its meaning".

Frames, introduced by Minsky in 1975 [118] represent another important development in Knowledge Representation. A frame usually represents a concept (or class) with associated attributes, which represent the properties shared by all instances of the class. An attribute is specified through the definition of a *slot*, which contains all information relevant to the attribute: restrictions on the number of possible values (called slot *fillers*), a default value, which is the one to take for the attribute if more specific information is missing, procedures for calculating the value when it is requested but not yet available, or procedures that are activated if the value is modified or deleted. Additionally, each slot has an associated domain for its fillers. Such domain can either be a concrete domain, such as strings or integers, or another frame specified through its identifier. Moreover it is possible to specify that a frame is a sub-frame of another one and therefore should inherit all of its properties, i.e. all of its slots.

A frame can also represent a single object of the domain, in which case it has a special attribute that relates it to the frame representing the class of which it is an instance. The slots of a frame representing an object are inherited by the frame representing its class, and to each slot a concrete value taken from its definition domain is associated. If not explicitly overridden, the value taken is the default value for the slot.

Reasoning in frame systems involves usually both the intensional and the extensional knowledge contained in the frame knowledge base. At the intensional level, the deduction process leads to the construction of a frame taxonomy, using both the explicit inclusion assertions in the knowledge base, and the structural information associated to the frame by virtue of their slots. The taxonomy induces a modification of properties at the extensional level, by assigning values to slots and by propagating the effects caused by the activation of the procedures associated to the slots.

Due to the lack of distinction between intensional and extensional knowledge and the presence of procedural attachments, however, the early frame based systems [27, 78] suffered from problems similar to those of Semantic Networks. Additionally, the treatment of default values lacked a clear semantics and the propagation of side effects made it difficult to predict the behaviour of the system, which in any case was dependent on the implementation.

1.2 Description Logics

Already Hayes [91] notes, that many aspects of Semantic Networks and Frames admit a translation in first order logic. Building on this correspondence, subsequent research in Knowledge Representation was

therefore aimed at overcoming the problems inherent in the early representation formalisms by formalizing their principal characteristics.

The first system with precisely this objective was KL-ONE[35], developed from the Ph.D. thesis of Brachman. The high expressivity of the representation language used in this system makes it however impossible to develop reasoning procedures that are sound and complete with respect to the formal semantics. It was in fact show later [142] that it is already undecidable to check whether a class expressed in the KL-ONE language contains an inconsistency that forces it to be necessarily empty. Successively, building on KL-ONE, a variety of knowledge representation systems, called *terminological* (or *concept based*) systems have been developed. These are all equipped with reasoning procedures that conform to the semantics. The most important of these systems are KANDOR [125], KRYPTON [34], NIKL [101], LOOM [114], CLASSIC [29], BACK [129], and KRIS [14].

Each terminological system is based on a specific *Description Logic* (this is the most popular denomination today, synonyms are *Concept Language* and *Terminological Logic*). The basic elements of a Description Logic are *concepts*, which denote classes, and *roles*, which are used to specify the properties of classes. Each concepts is interpreted as a subset of the interpretation domain (the set of *instances* of the concept), while each role is interpreted as a binary relation on that domain. A Description Logic is characterized by the set of constructors that can be used to build complex concept and role expressions starting from a set of concept and role names. The most common concept constructors include constructors that correspond to the boolean connectives and constructors that allow one to quantify over all domain elements connected through a certain role. For example, the concept expression $\texttt{Course} \sqcap \neg \texttt{AdvCourse} \sqcap \forall \texttt{followedby.GradStud}$ denotes the class of courses that are not advanced and that are followed only by graduate students. The symbol "$\sqcap$" denotes conjunction and is interpreted as set intersection, while "$\neg$" denotes negation and is interpreted as complement with respect to the domain of interpretation. The expression $\forall \texttt{followedby.GradStud}$ denotes the set of objects that are connected through role `followedby` only to instances of `GradStud`. An expression of the form $\forall R.C$, where R is a role and C is a concept is called *universal quantification over roles*, and is used to specify the properties of all objects connected by means of a role. The expression $\exists \texttt{followedby.Student}$ denotes all objects that are connected by means of `followedby` to at least one instance of `Student`. Expressions of the form $\exists R.C$ are called *existential quantifications*, and can appear also in the unqualified form $\exists R$. In the unqualified case it is only required that an object connected through role R exists, without specifying further properties that this object must satisfy.

An additional constructor demonstrates particularly useful. It stems from the cardinality restrictions originally present in frame bases systems, and is called *number restrictions*. In Description Logics, number restrictions are denoted by $(\geq n\, R)$ and $(\leq n\, R)$, where n is a nonnegative integer (in the formalism we are going to introduce, and which is in fact a variant of Description Logics, we use the notation $\exists^{\geq n} R$ and $\exists^{\leq n} R$ instead). They allow one to pose restrictions on the number of times an object can appear as a component in a role R. As an example, the courses followed by no more that 10 students can be represented by means of $\texttt{Course} \sqcap (\leq 10\, \texttt{followedby})$.

Many of the Description Logics studied so far have either no role constructors at all or consider only the possibility to use *role intersection* (see [65] for a recent systematic presentation of all the constructors mentioned so far). A notable increase in expressivity is achieved by allowing one to refer to the *inverse* of a role. As an example, above, it would probably be more correct to use $\texttt{follows}^{-}$ in place of `followedby`. The inverse constructor has been rarely considered till now (an exception is [52]), mainly because it is problematic to handle it by the traditional reasoning methods developed for Description Logics, which are based on tableau like calculi. This is especially true in combination with number restrictions and cyclic assertions (see below).

Starting from a Description Logic one can then build *terminological knowledge bases*. These consist usually of two separate components, an intensional and an extensional one. The *extensional* part contains assertions about real world objects, specifying the concepts they are instances of and the other objects to which they are connected through roles. The *intensional* part, instead, contains general assertions about the concepts used in the extensional part, and specifies their properties and their mutual relations. It is constituted by a set of *assertions* that are of two types: *Primitive concept specifications*, which have the form $C \doteq E$, are used to specify by means of a concept expression E necessary conditions for an object of the domain to be an instance of a concept C. *Concept definitions*, instead, which have the form $C \doteq E$, allow one to specify (by means of E) necessary and sufficient conditions for an object to be an instance of the concept C to be defined. An assertion is satisfied in an interpretation, if the extensions of the concept and

the concept expressions satisfy the set inclusion or equality. A *model* of (the intensional part) of a knowledge base is an interpretation that satisfies all assertions in the knowledge base.

The basic inference tasks to be carried out on the intensional part of a knowledge base are:

1. *Knowledge base consistency*: Does the knowledge base admit a model? This is already meaningful if the extensional part is empty, since the intensional part alone may already contain implicit contradictions.

2. *Concept consistency*: Given a knowledge base and a concept, is there a model of the knowledge base in which the concept has a nonempty extension? This is important in the design phase of a knowledge base, in order to eliminate meaningless parts.

3. *Concept subsumption*: Given a knowledge base and two concepts, is the extension of one concept a subset of the extension of the other one in every model of the knowledge base? Subsumption allows one to detect dependencies that are implicit in the knowledge base. It is at the basis of *classification*, the typical form of reasoning performed in terminological systems, that consists in computing the taxonomy of all concepts in the knowledge base. The taxonomy can then be used both to answer queries at the intensional level [149], and to optimize the query answering process [24].

When considering the combination of both intensional and extensional part, several other reasoning tasks arise, such as checking if the intensional and extensional part of a knowledge base are consistent with each other, verifying if an object is necessarily an instance of a concept in all models of a knowledge base (*instance checking*), or finding all instances of a concept in a knowledge base (*retrieval*). In this work we concentrate on intensional reasoning only. A detailed treatment of the issues related to extensional reasoning can be found in [12, 139, 69, 138].

Description Logics have been studied intensively, both with respect to expressivity and with respect to the complexity of reasoning. These two aspects influence each other, and as pointed out in a seminal paper by Brachman and Levesque [32], an even seemingly small change in expressivity can result in a big change in computational complexity of reasoning. The issue consists therefore in finding the "right" balance between these two aspects.

Initially, research concentrated on the restricted problem of *pure subsumption*, which is subsumption between concept expressions assuming an empty knowledge base. The study of this problem was motivated by the following observation: Under the assumption that the assertions in the knowledge base contain no cyclic references between concepts, the problem of concept subsumption with respect to such a knowledge base can be reduced to pure subsumption. This is achieved simply by unraveling all concept definitions and specifications. The assumption of acyclicity is common to many terminological systems, and although the unraveling of the definitions can in the worst case lead to an exponential increase in the size of the concept expressions this seems not to happen in practical cases under reasonable assumptions [121]. The computational complexity of pure subsumption has been exactly characterized for almost all interesting combinations of constructs [119, 64, 143, 61, 65], and this problem can now be regarded as settled. The computational complexity ranges from polynomially solvable for the simplest languages (containing only conjunction, universal quantification and unqualified existential quantification) to **NP**-complete (e.g. if we add qualified existential quantification), **coNP**-complete (e.g. if we add disjunction), or **PSPACE**-complete (if we add both).

On the other hand, the problem of reasoning while taking into account the assertions of a knowledge base is not so well studied and far from being completely characterized in all its sub-cases. This holds in particular if cyclic assertions are allowed, although several results have already been obtained [9, 122, 11, 38, 36, 52] (see also [70] for a recent survey covering also some of these aspects). First of all, while for acyclic knowledge bases, only one semantics can be adopted, different possibilities arise, if we have a cyclic knowledge base [122]. In fact, there is still no agreement, on which is the most appropriate semantic specification for cycles, and there have also been proposals for an integration of the various types [55, 141]. In this thesis we adopt *descriptive semantics*, which accepts as models all interpretations that satisfy the cyclic assertions (as opposed to fixpoint semantics, which accept as models only particular interpretations satisfying the assertions). Additionally, the presence of cyclic assertions greatly increases the complexity of reasoning [10, 11, 122]. For these reasons, actual terminological systems (with the exception of K-REP [115]) rule out the use of cycles in the knowledge base. Their importance in practice is however beyond dispute, since many practical concepts can be expressed only by using cyclic assertions, and cyclic assertions are used anyway in other representation formalisms.

Cyclic assertions are especially problematic in the presence of constructors for number restrictions (a restricted form called *functionality* is sufficient for causing problems) and inverse roles. In this case the resulting knowledge base may be consistent, but only if we admit an infinite interpretation domain. This phenomenon is new in terminological representation systems and has not been dealt with before. It requires a distinction between reasoning in finite models and reasoning in unrestricted models. The development of reasoning techniques for both cases and the characterization of their computational complexity constitutes the most important contribution of this thesis.

1.3 Semantic Data Models

As already mentioned, in parallel to the work done in Knowledge Representation, also in Databases formalisms for the representation of complex data relationships have been studied. Together with the relational model [47], in the '70ies so called *semantic data models* have been introduced [45, 103, 98]. The relational model, which allows the database designer to separate the logical design of a database from its implementation, is at the basis of all recent industrial database applications. Conceptual and semantic data models were initially developed with a different objective: to be used in the design phase of database applications as a tool for conceptual modeling. Indeed, they put their emphasis on the correct modeling of the relations that exist between data, and offer sophisticated structuring primitives. This allows the designer to represent the data in a form that is conceptually similar to the way in which this data is effectively used in the real world. Additionally, they favor a top-down development of the schema that increases modularity and therefore simplifies the design and the successive interaction with the database. Recently there have also been attempts at basing entire database management systems on conceptual models.

The first semantic data model was proposed by Abrial in 1974 [5]. Successive research has led to the development of a great variety of semantic data models with different characteristics [98]. One of the best known and widely used in industrial applications is the *Entity-Relationship* (ER) model introduced by Chen in [45]. The basic concepts of the ER model are *entities*, *relationships*, and *attributes*. Entities denote classes of objects with common properties. These are modeled by attributes, which represent atomic properties and have an associated domain, and by the participation of the entity in relationships, via so called *roles*. The relationships can be of arbitrary arity, and can also have associated attributes. The distinction between entities and relationships, however, is strict, and "higher order" relations, i.e. relations over relationships, are not allowed. By means of *cardinality constraints* between an entity and a relationship, associated to the corresponding role that represents the participation, it is possible to limit (from below and from above) the number of times that each instance of the entity can participate in the relationship. Finally, in the ER model, one can assert so called *ISA* relations between entities, which are interpreted as set inclusions between the respective classes. In such a way an entity inherits all properties of the entities to which it is connected via ISA relations.

In restricted forms of the ER model, where the use of ISA relations is forbidden, it is commonly assumed that the sets of instances of different entities are disjoint (see for example [111]). In the presence of ISA, however, this assumption does not make sense. The basic ER model does not provide means for expressing explicit disjointness of entities or the fact that an entity represents the union of other entities, although recent papers stress the importance of such constructs in database specification [42, 48]. Therefore extensions have been proposed in which inclusion between entities can be expressed not only by means of ISA relations but also by so called *generalization hierarchies* (see for example [20]). They are used whenever an entity represents the generalization of a set of other entities, and the set of its instances is the union of the sets of instances of the entities it generalizes. Additionally, the most specific entities in the hierarchy are assumed to be pairwise disjoint. Therefore generalization hierarchies combine disjointness and explicit disjunction.

Reasoning in the ER model includes verifying if an entity is satisfiable and reasoning on inheritance. Satisfiability of an entity amounts to verifying whether the entity can be populated in a database that satisfies all constraints imposed by the schema [7, 111]. Deducing inheritance amounts to verifying whether the schema forces all instances of a certain entity to be also instances of another entity. This is clearly the case if the two entities are connected by a chain of ISA relations or one entity is below the other in a generalization hierarchy. It may arise, however, also due to the interaction of cardinality constraints along a cycle in the schema, and the requirement that the database be finite.

In fact, unlike in terminological systems, the common assumption in Databases is that a database is finite. Reasoning with respect to finite structures is different from reasoning with respect to unrestricted

structures [50], and this holds already for the ER model [111].

1.4 Object-Oriented Data Models

Object-oriented data models have been proposed recently with the aim of integrating formalisms used in the design of databases with the modern programming languages based on the object-oriented paradigm [108, 21, 97, 104]. In contrast to traditional data models, that are value oriented, in object-oriented models the emphasis is on the complex structure that objects of the domain to be modeled can have. Each object is identified uniquely by an *object identifier* and has a possibly complex structure. This structure is determined by the classes the object belongs to, and these classes are specified through inheritance and typing mechanisms.

A type is specified by applying inductively a set of type constructors starting from a set of basic types and classes. All models considered in the literature have constructors for *aggregation* and *grouping*, to build tuples and sets, respectively. An additional constructor that has been considered, is that of *union* [1]. The complex types resulting from the repeated application of constructors are then used for the definition of classes, and determine the structure of the objects that are instances of the classes. Usually, an object-oriented schema is interpreted over a finite structured domain. Formally, this is achieved by assigning to each object of the domain a value, whose structure is constructed in a way similar to that of the complex types. Therefore, we have values that have the structure of a tuple or a set and are composed of values with a simpler structure. By means of inclusion assertions between classes it is then possible that a class inherits properties from its subclasses.

The semantics of an object-oriented schema is specified by assigning to each object of the interpretation domain a structured value, and specifying the classes an object belongs to. In every database corresponding to the schema, the set of objects that constitute the instances of the classes must satisfy the respective inclusion assertion. Moreover, the value assigned to an instance of the class must be compatible with the structure of the type associated to the class.

Reasoning in object-oriented model includes subtyping, which consists of verifying whether a type is a subtype of another type in every database satisfying the constraints imposed by the schema, and in type consistency, which consists in checking whether a type is consistent in a schema.

An important aspect of object-oriented models is the possibility to specify by means of so called *methods* dynamic aspects of classes, i.e. aspects related to the evolution and manipulation of the database. A method is a procedure associated to a class and which can operate on the instances of the class in the database, in order to perform updates or deletions. Methods are an active area of research, but still resist to a satisfactory formalization [105, 21, 22, 3]. In the present work we do not deal with methods and concentrate only of the structural component of object-oriented schemata.

1.5 Goal of the Thesis and Main Results

The thesis is concerned with reasoning on formalisms for the representation of structured knowledge, by (1) studying its intrinsic complexity, (2) designing effective techniques for performing deduction at the intensional level, and (3) analyzing the computational properties of the developed methods. The study is performed in the context of a formal framework that allows us to abstract with respect to the peculiarities of the different contexts.

We first devise in an incremental manner a set of abstract representation languages, called $\mathcal{L}$*-languages*. In $\mathcal{L}$-languages one can build (by means of certain constructors that characterize the language and similar to what is done in Description Logics) complex class and attribute expressions. These can then be used to define $\mathcal{L}$*-schemata* which represent knowledge at the intensional level. The meaningful constructors we consider in this thesis are the following: the propositional connectives on classes including union, number restrictions, which generalize functional and existence constraints in Databases, inverse attributes and more complex traversal patterns obtained by constructing arbitrary regular expressions of attributes. We introduce also a new constructor, called "repeat", not considered before in formalisms for structured knowledge representation. By means of the repeat constructor together with cyclic definitions of classes, it becomes possible to define several inductive structures such as lists, trees, etc.. We consider different types of schemata,

including the most general ones without any restriction on the form of the assertions that constitute the schema.

We first show that $\mathcal{L}$-languages and $\mathcal{L}$-schemata are indeed adequate for our purposes, since they subsume (in their more expressive variants) the structural aspects of the most common representation formalisms used both in Knowledge Representation and in Databases, namely frame systems, and semantic and object-oriented data models. This is done by providing translations that preserve the results of deductions from all these formalism to $\mathcal{L}$-schemata. These translations show that for this purpose we need the combination of inverse attributes, disjunction, and number restrictions, together with cyclic class specifications, and that this is also sufficient for our purposes (primitive $\mathcal{LUNI}$-schemata).

The main part of the thesis is concerned with the design of effective techniques for intensional reasoning on $\mathcal{L}$-schemata and the study of their computational properties. The reasoning tasks we consider are the ones we already mentioned in the context of Description Logics, namely *schema consistency* ("Is there a nonempty database that satisfies all constraints expressed in the schema?"), *class consistency* ("Can a class be populated in a database satisfying the schema?"), and *class subsumption* ("Does a class denote a subset of another class in every database satisfying the schema?").

As a first step we analyze the intrinsic complexity of these reasoning tasks, and show that even for the least expressive $\mathcal{L}$-language and the simplest form of schemata, which are subsumed by most known representation formalisms, reasoning is already intractable, and more precisely **coNP**-complete. Adding either the possibility to express disjunction or the possibility to use cycles in the schema, class consistency becomes even **PSPACE**-hard. We then show that the combination of both features makes reasoning even **EXPTIME**-hard, i.e. as hard as reasoning on the most general type of schemata using the most expressive language. This is done by introducing a general technique for the elimination of qualified existential quantification from a schema. This technique allows us to establish further complexity results and to characterize completely the computational complexity of reasoning on (almost) all types of $\mathcal{L}$-schemata.

The interaction that occurs between some of the constructs (namely inverse attributes and number restrictions) when they are used inside cyclic schemata causes the finite model property to fail to hold. This means that reasoning differs in the case where we consider arbitrary models (i.e. databases of arbitrary, possibly infinite size) from the case where we restrict our attention to finite ones. The latter assumption is in fact the common one in Databases, while finiteness of the underlying domain has seldom been an issue in Knowledge Representation. For this reason we analyze reasoning in both contexts separately.

In the case of unrestricted models the reasoning techniques developed in Knowledge Representation are adequate. They are based on the similarity that exists between the interpretative structures of $\mathcal{L}$-languages and those of Propositional Dynamic Logics (PDLs), which are formalisms specifically designed for reasoning about program schemes. Such similarity results in a well-known correspondence between both formalisms which we extend in this work in order to cope with all the constructs present in $\mathcal{L}$-languages. Motivated by the importance of primitive $\mathcal{LUNI}$-schemata, we concentrate first on this case, and show that known reasoning procedures can be applied after performing an easy transformation on the schema. This is not the case anymore for the full featured language, where we first need to perform quite intricate transformations on the schema to obtain a translation in PDL. In order to establish decidability of the resulting PDL we use techniques based on automata on infinite objects, a tool extensively used for establishing upper bounds for several modal logics of programs. These techniques are based on the fundamental *tree model property* shared by all these formalism. It states that every satisfiable formula admits particular models that have the form of an (infinite) tree of bounded degree in which the formula holds at the root. Satisfiability of a formula can therefore be reduced to the problem of nonemptiness of the finite state tree automaton that accepts precisely the tree-like models of the formula. We show that such a reduction can also be carried out for the PDL at hand, thus extending recent results on the decidability of very expressive PDLs. By exploiting decidability and complexity results for nonemptiness of particular types of tree automata, we obtain in fact a tight upper bound for reasoning on the most general form of schemata expressed in an extremely rich language, showing the optimality of the resulting method.

In the case of finite models, on the other hand, the known reasoning methods cannot be used, since the existence of tree-like models is not guaranteed if the domain has to be finite. For this reason completely new reasoning techniques are developed. They are based on the idea of separating the reasoning process in two distinct phases. The first phase deals with all constructors except number restrictions and existential quantifications (which in the unqualified form is just a particular case of number restriction), and builds an expanded schema in which these constructors are embedded implicitly in the classes and attributes.

In the second phase the assertions involving number restrictions are used to derive from this expanded schema a system of linear inequalities. The system is defined in such a way that its solutions of a certain type (acceptable solutions) are directly related to the finite models of the original schema. In particular, from each acceptable solution one can directly deduce the cardinalities of the extensions of all classes and attributes in a possible finite model. The proposed method allows us to establish decidability in worst case deterministic double exponential time of finite model reasoning in free $\mathcal{LCNI}$-schemata, and provides a tight upper bound for the important case of primitive $\mathcal{LUNI}$-schemata, which we again analyze separately. We study also the complexity of the proposed method under specific assumptions, and it turns out that under conditions that are very common in Databases the algorithm may exhibit an efficient behaviour.

1.6 Structure of the Thesis

The rest of the thesis is organized as follows:

- In Chapter 2 we provide a formal introduction to $\mathcal{L}$-languages, their use in the construction of $\mathcal{L}$-schemata, and the reasoning services that are considered on $\mathcal{L}$-schemata.
- In Chapter 3 we present the translations from frames, semantic data models, and object-oriented data models to $\mathcal{L}$-schemata, and of the associated reasoning services to reasoning services on $\mathcal{L}$-schemata.
- In Chapter 4 we prove various lower bounds for reasoning on $\mathcal{L}$-schemata. In particular, we show that for the simplest language considered and the simplest form of schemata, verifying if a class is consistent in a schema is already **coNP**-complete. The task becomes **PSPACE**-complete if the schema may contain either union or cycles, and even **EXPTIME**-complete if we allow for both. We show also that various constructs present in $\mathcal{L}$-languages are problematic and that their unrestricted use leads to undecidability.
- In Chapter 5 we analyze unrestricted model reasoning on $\mathcal{L}$-schemata, considering first the simpler case of primitive $\mathcal{LUNI}$-schemata and then the general case of the most expressive $\mathcal{L}$-language. The decidability and complexity results are established by exploiting the correspondence between $\mathcal{L}$-languages and PDLs.
- In Chapter 6 we analyze finite model reasoning on $\mathcal{L}$-schemata. We first present a technique that solves class consistency in primitive $\mathcal{LUNI}$-schemata in deterministic exponential time, which gives us a tight complexity bound. An analysis of the proposed method shows that it behaves efficiently under reasonable assumptions that are very common in Databases. We successively generalize the technique to arbitrary schemata and general negation, obtaining a deterministic double exponential upper bound for this case.

In order to make the thesis self-contained, we include also two appendixes containing additional notions that are required for the comprehension of the material in Chapter 5.

- Appendix A contains the basic notions about finite automata on infinite strings and infinite trees, a formal tool extensively used to establish decidability and lower bounds for various modal and program logics, and in particular for PDLs.
- Appendix B gives a brief introduction to Propositional Dynamic Logics, their variants used in this thesis, and describes the correspondence between $\mathcal{L}$-languages and PDLs.

Chapter 2

Representing Intensional Knowledge by $\mathcal{L}$-Schemata

In this chapter we introduce $\mathcal{L}$-languages which are an abstract formalism that allows for the representation of intensional knowledge in the form of $\mathcal{L}$-schemata. $\mathcal{L}$-languages permit to build complex class and attribute expressions by applying suitable constructors starting from a set of atomic symbols. Such expressions are then used in assertions to construct $\mathcal{L}$-schemata.

The syntax we use for $\mathcal{L}$-languages is in the style of Description Logics (see for example [70]). In Section 2.1 we present syntax and semantics of $\mathcal{L}$-languages, and in Section 2.2 we describe how to build $\mathcal{L}$-schemata. In Section 2.3 we discuss the issues related to reasoning on schemata.

2.1 Syntax and Semantics of $\mathcal{L}$-Languages

An *$\mathcal{L}$-language* is composed by a set of symbols taken from the following disjoint countable alphabets:

- **CN**: the alphabet of *class names*,
- **AN**: the alphabet of *attribute names*.

A class name represents an *atomic class expression*, and an attribute represents an *atomic link expression*. Starting from these atomic expressions more complex *class expressions* and *link expressions* are constructed by applying suitable *constructors*. Each $\mathcal{L}$-language is characterized by the set of constructors that can be used. The constructors we consider in this thesis have been regarded as important in the context of both Knowledge Representation and conceptual database modeling [35, 70, 25, 43]. We denote with $\mathcal{L}_0$ the core language containing only the basic constructors which are present in all $\mathcal{L}$-languages. Each additional constructor is denoted by an identifying letter, and we use the convention of naming a language that includes certain constructors (in addition to those of $\mathcal{L}_0$) by a string that starts with $\mathcal{L}$ and includes the corresponding letters. Class and link expressions constructed using only constructors of a language $\mathcal{L}$ are called *$\mathcal{L}$-expressions*.

We denote class names with the letter C, and attribute names with A, possibly with subscripts. Arbitrary class expressions are denoted with E and arbitrary link expression with L.

The semantics for $\mathcal{L}$-languages is given through the notion of interpretation. An interpretation $\mathcal{I} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty set $\mathcal{I}$, called the *domain* of the interpretation, and an *interpretation function* $\cdot^{\mathcal{I}}$, which maps each class expression E to a subset $E^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, called the *extension* of E, and each link expression L to a binary relation $L^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$, called the *extension* of L. The elements of $E^{\mathcal{I}}$ and $L^{\mathcal{I}}$ are called *instances* of E and L respectively. Each constructor poses specific conditions that the interpretation function must satisfy. We say that a class or link expression X is *consistent* if it admits an interpretation $\mathcal{I}$ in which its extension $X^{\mathcal{I}}$ is nonempty, and we say that X is *inconsistent* otherwise.

2.1.1 The Core Language ($\mathcal{L}_0$, $\mathcal{L}^-$)

The core language we consider, called $\mathcal{L}_0$, allows one to build class expressions according to the following syntax:

$$\begin{array}{lcl} E & \longrightarrow & \top \mid \bot \mid C \mid \neg C \mid E_1 \sqcap E_2 \mid \forall L.E \mid \exists L \\ L & \longrightarrow & A. \end{array}$$

Intuitively, $\top$ represents the whole domain, and $\bot$ the empty set. $\neg C$ represents the *negation* of a class name, and is interpreted as the complement with respect to the domain of interpretation. $E_1 \sqcap E_2$ represents the *conjunction* of two class expressions and is interpreted as set intersection. $\forall L.E$ is called *universal quantification* and is used to denote those objects of the interpretation domain that are connected through link L only to instances of the class E. Similarly, $\exists L$ is called *(unqualified) existential quantification*, and is interpreted as those objects that are connected through link L to at least another object of the domain. More formally, every interpretation $\mathcal{I}$ satisfies the following conditions:

$$\begin{array}{rcl} C^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\ A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\ \top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\ (\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\ (E_1 \sqcap E_2)^{\mathcal{I}} & = & E_1^{\mathcal{I}} \cap E_2^{\mathcal{I}} \\ (\forall L.E)^{\mathcal{I}} & = & \{o \in \Delta^{\mathcal{I}} \mid \forall o' \colon (o,o') \in L^{\mathcal{I}} \rightarrow o' \in E^{\mathcal{I}}\} \\ (\exists L)^{\mathcal{I}} & = & \{o \in \Delta^{\mathcal{I}} \mid \exists o' \colon (o,o') \in L^{\mathcal{I}}\}. \end{array}$$

Notice that $\mathcal{L}_0$ contains no rules for defining complex link expressions, and therefore the only allowed links are attribute names. However, this language has already the basic features that allow for describing classes with interesting properties: Universal quantification is used for assigning a type to an attribute, which is a basic modeling feature present in all data models and knowledge representation systems [98, 108, 35, 29, 31]. Existential quantification permits to express totality of attributes and is also a means to express incomplete information [110]. Additionally, through conjunction one can express multiple inheritance of properties, while negation can be used in schemata to force disjointness of classes [7, 59].

Example 2.1.1 The following $\mathcal{L}_0$-class expression could be interpreted as a set of basic university courses:

$$\begin{array}{l} \neg\texttt{AdvCourse} \sqcap \\ \exists\texttt{taughtby} \sqcap \forall\texttt{taughtby.Teacher} \sqcap \\ \exists\texttt{followedby} \sqcap \forall\texttt{followedby.Student}. \end{array}$$

Its instances are not instances of the class `AdvCourse`, are connected through the attributes `taughtby` and `followedby` only to instances of the classes `Teacher` and `Student`, respectively, and at least to one such instance for `taughtby` and `followedby`. This expresses that basic courses are disjoint from advanced ones, that each course is taught and followed by someone, and that it is taught only by teachers and followed only by students. ∎

In this thesis we consider also the language $\mathcal{L}^-$, which is obtained from $\mathcal{L}_0$ by dropping "$\bot$" and the constructor for negation. It is easy to see that this makes it impossible to express classes in $\mathcal{L}^-$ that necessarily have an empty extension in all interpretations.

Proposition 2.1.2 *Every $\mathcal{L}^-$-class expression is consistent.*

Proof. For an $\mathcal{L}^-$-class expression E we define the following interpretation $\mathcal{I}' := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. We set $\Delta^{\mathcal{I}} := \{o\}$, for any class name C appearing in E we set $C^{\mathcal{I}} := \{o\}$, and for any attribute name A appearing in E we set $A^{\mathcal{I}} := \{(o,o)\}$. It is easy to verify by induction on the structure of E that $E^{\mathcal{I}} = \{o\}$. □

2.1.2 Disjunction ($\mathcal{LU}$)

The language $\mathcal{LU}$ is obtained from $\mathcal{L}_0$ by adding the constructor $\mathcal{U}$ for expressing *disjunction* of classes. The additional syntax rule is:

$$E \longrightarrow E_1 \sqcup E_2.$$

The class expression $E_1 \sqcup E_2$ is interpreted as the union of the extensions of the classes E_1 and E_2. Formally, every interpretation $\mathcal{I}$ satisfies the condition:

$$(E_1 \sqcup E_2)^{\mathcal{I}} = E_1^{\mathcal{I}} \cup E_2^{\mathcal{I}}.$$

Note that, since negation can be applied only to class names, disjunction cannot be expressed by De Morgan's laws using conjunction and negation. By means of disjunction, which has been considered both in database models and Knowledge Representation, incomplete information can be represented [110]. It allows one also to express covering constraints [109] and generalization ISA relationships [154].

Example 2.1.1 (cont.) Using disjunction we can refine our class expression denoting basic courses and express that they are taught either by professors or graduate students:

$$\begin{array}{l}\neg\texttt{AdvCourse} \sqcap \\ \exists\texttt{taughtby} \sqcap \forall\texttt{taughtby.}(\texttt{Professor} \sqcup \texttt{GradStud}) \sqcap \\ \exists\texttt{followedby} \sqcap \forall\texttt{followedby.Student}.\end{array}$$

■

When the language we consider includes disjunction and negation of class names, we may regard $\top$ as an abbreviation of $C \sqcup \neg C$, where C is any class name.

2.1.3 Qualified Existential Quantification ($\mathcal{LE}$)

The basic language $\mathcal{L}_0$ allows one to express only an unqualified form of existential quantification since using the constructor $\exists L$ does not permit to impose any conditions on the objects connected through link L. The language $\mathcal{LE}$ is obtained by removing this limitation and allowing one to express the existence of an object connected through a link and satisfying certain conditions. The constructor $\mathcal{E}$, called *qualified existential quantification* is described by the following syntax rule:

$$E \longrightarrow \exists L.E.$$

It has been studied mainly in the context of structured knowledge representation languages [34, 62].

The expression $\exists L.E$ is interpreted as the set of objects that are connected through the link L to at least one object that is an instance of E. Formally, every interpretation $\mathcal{I}$ satisfies the condition:

$$(\exists L.E)^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} \mid \exists o' \colon (o, o') \in L^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}}\}.$$

Example 2.1.1 (cont.) Qualified existential quantification can be used to express the property of advanced courses to be followed by at least one graduate student.

$$\begin{array}{l}\exists\texttt{taughtby} \sqcap \forall\texttt{taughtby.Teacher} \sqcap \\ \forall\texttt{followedby.Student} \sqcap \exists\texttt{followedby.GradStud}.\end{array}$$

■

2.1.4 Number Restrictions ($\mathcal{LN}$, $\mathcal{LF}$, $\mathcal{LQ}$)

The language $\mathcal{LN}$ is obtained from $\mathcal{L}_0$ by adding *number restrictions*, which are denoted by $\mathcal{N}$. They allow one to express bounds on the number of objects that are connected through an attribute to a certain object. Syntactically a class expression in $\mathcal{LN}$ is obtained by adding to the syntax rules for $\mathcal{L}_0$ the following, where m is a positive and n a nonnegative integer:

$$E \longrightarrow \exists^{\geq m} L \mid \exists^{\leq n} L.$$

An instance of $\exists^{\geq m} L$ is connected through link L to at least m distinct objects, and similarly, an instance of $\exists^{\leq n} L$ is connected through link L to at most n distinct objects. Formally, every interpretation $\mathcal{I}$ satisfies the condition:

$$\begin{aligned} (\exists^{\geq m} L)^{\mathcal{I}} &= \{o \in \Delta^{\mathcal{I}} \mid \sharp\{o' \mid (o, o') \in L^{\mathcal{I}}\} \geq m\} \\ (\exists^{\leq n} L)^{\mathcal{I}} &= \{o \in \Delta^{\mathcal{I}} \mid \sharp\{o' \mid (o, o') \in L^{\mathcal{I}}\} \leq n\}, \end{aligned}$$

where for a set S, $\sharp S$ denotes the cardinality of S.

Example 2.1.1 (cont.) We can now pose restrictions on the number of students that should follow a basic course and require that each course be taught by exactly one teacher, and followed by at least three and at most 100 students:

$$\begin{aligned} &\neg\texttt{AdvCourse} \sqcap \\ &\forall\texttt{taughtby.Teacher} \sqcap \exists^{\geq 1}\texttt{taughtby} \sqcap \exists^{\leq 1}\texttt{taughtby} \sqcap \\ &\forall\texttt{followedby.Student} \sqcap \exists^{\geq 3}\texttt{followedby} \sqcap \exists^{\leq 100}\texttt{followedby}. \end{aligned}$$

■

Although number restrictions appear in various forms in most conceptual and semantic database models [5, 84, 155], they are rarely present in the object-oriented setting (an exception is [29]). It is worth noting that the use of number restrictions allows us to represent several forms of existence and functional dependencies, and is very common in structured knowledge representation systems, as pointed out, for example, in [78, 31].

We use the letter $\mathcal{F}$ to denote the limited form of number restriction in which the only number allowed is 1. Observe that since a number restriction of the form $\exists^{\geq 1} L$ is equivalent to the existential quantification $\exists L$, the language $\mathcal{LF}$ is obtained from $\mathcal{L}_0$ by just adding the constructor $\exists^{\leq 1} L$, which is called a *functional restriction*. Such a constructor is often implicit in object-oriented data models, where attributes are required to be single-valued.

Qualified number restriction [94], denoted by $\mathcal{Q}$, are a generalization of number restrictions, which both pose restrictions on the number of objects connected through a link and also allow to specify that only objects that are instances of a certain class should be counted. They are introduced via the following syntax rule:

$$E \longrightarrow \exists^{\geq m} L.E \mid \exists^{\leq n} L.E,$$

and are interpreted by requiring that every interpretation $\mathcal{I}$ satisfies the following conditions:

$$\begin{aligned} (\exists^{\geq m} L.E)^{\mathcal{I}} &= \{o \in \Delta^{\mathcal{I}} \mid \sharp\{o' \mid (o, o') \in L^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}}\} \geq m\} \\ (\exists^{\leq n} L.E)^{\mathcal{I}} &= \{o \in \Delta^{\mathcal{I}} \mid \sharp\{o' \mid (o, o') \in L^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}}\} \leq n\}. \end{aligned}$$

Example 2.1.1 (cont.) With qualified number restrictions we can express that each advanced course is followed by at least three graduate students, and at most five students that are not graduate.

$$\begin{aligned} &\forall\texttt{taughtby.Teacher} \sqcap \exists^{\geq 1}\texttt{taughtby} \sqcap \exists^{\leq 1}\texttt{taughtby} \sqcap \\ &\forall\texttt{followedby.Student} \sqcap \\ &\exists^{\geq 3}\texttt{followedby.GradStud} \sqcap \exists^{\leq 5}\texttt{followedby.}\neg\texttt{GradStud}. \end{aligned}$$

■

We make use of the following abbreviations:

$$\begin{aligned} \exists^{=m} L &:= \exists^{\geq m} L \sqcap \exists^{\leq m} L \\ \exists^{=m} L.E &:= \exists^{\geq m} L.E \sqcap \exists^{\leq m} L.E. \end{aligned}$$

2.1.5 Inverse Attributes ($\mathcal{LI}$)

The language $\mathcal{LI}$ is obtained from $\mathcal{L}_0$ by adding the constructor $\mathcal{I}$ on attributes which allows one to refer to the *inverse* of an attribute name. The additional syntax rule is:

$$L \longrightarrow A \mid A^{-}.$$

By an *atomic attribute* we mean either an attribute name or an attribute name to which the converse constructor is applied.

The inverse attribute A^{-} is interpreted as the inverse of the binary relation that gives the extension of A. Formally, every interpretation $\mathcal{I}$ satisfies the condition:

$$(A^{-})^{\mathcal{I}} = \{(o, o') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (o', o) \in A^{\mathcal{I}}\}.$$

Inverses of attributes are considered in functional data models [146] and are implicitly present in semantic data models where relations can be navigated in any direction. They have rarely been considered in the object-oriented framework, although their importance has been mentioned in [43], a recently proposed standard for object-oriented databases.

Example 2.1.1 (cont.) Using inverse attributes we can correctly model courses by using the attributes `follows` and `teaches`.

$$\begin{aligned} &\neg\texttt{AdvCourse} \sqcap \\ &\exists\texttt{teaches}^{-} \sqcap \forall\texttt{teaches}^{-}.\texttt{Teacher} \\ &\exists\texttt{follows}^{-} \sqcap \forall\texttt{follows}^{-}.\texttt{Student}. \end{aligned}$$

■

2.1.6 General Negation ($\mathcal{LC}$)

We consider the possibility of referring to the *negation* of an arbitrary class expression, which we denote by the letter $\mathcal{C}$, and which is interpreted as complement with respect to the domain of interpretation. This constructor does not increase expressivity for those languages that are closed under complement. For example, since disjunction and existential quantification are complementary to conjunction and universal quantification respectively, the language $\mathcal{LUE}$ is equivalent to $\mathcal{LUEC}$ which is equivalent to $\mathcal{LC}$. A similar consideration holds if we add (qualified) number restrictions to these languages, since the two operators $\exists^{\geq m} L$ and $\exists^{\leq n} L$ are complementary to each other.

We consider also a transformation of class expressions involving general negations into an equivalent form in which negation is pushed inside the other constructors as much as possible.

Definition 2.1.3 The *negation normal form* of an expression $\neg E$, denoted $\sim E$, is defined as follows:

$$\begin{aligned} \sim C &:= \neg C \\ \sim(\neg E) &:= \begin{cases} C, & \text{if } E \text{ is a class name } C \\ \sim(\sim E), & \text{otherwise} \end{cases} \\ \sim(E_1 \sqcap E_2) &:= \sim E_1 \sqcup \sim E_2 \\ \sim(E_1 \sqcup E_2) &:= \sim E_1 \sqcap \sim E_2 \\ \sim\forall L.E &:= \exists L.\sim E \\ \sim\exists L.E &:= \forall L.\sim E \\ \sim\exists^{\geq m} L &:= \exists^{\leq m-1} L \\ \sim\exists^{\leq n} L &:= \exists^{\geq n+1} L \\ \sim\exists^{\geq m} L.E &:= \exists^{\leq m-1} L.E \\ \sim\exists^{\leq n} L.E &:= \exists^{\geq n+1} L.E. \end{aligned}$$

■

It follows from the semantics of the constructors that for any interpretation $\mathcal{I}$ and for any class expression E, $(\neg E)^{\mathcal{I}} = (\sim E)^{\mathcal{I}}$.

2.1.7 Arbitrary Links ($\mathcal{LL}$, $\mathcal{LD}$, $\mathcal{L}\Delta$, $\mathcal{LV}$)

We have introduced so far powerful constructors for the generation of complex class expressions, while the expressivity of attributes is still somewhat limited. In fact, when using universal and existential quantification and number restrictions to refer to instances of other classes we are constrained to refer only to the objects that are directly connected via some attribute (or inverse attribute). To overcome this limitation we introduce now constructors for building more complex links, and for using these in the definition of new types of class expressions. The most intuitive approach in this direction is to use as links regular expressions over atomic attributes.

The language $\mathcal{LL}$ is obtained from $\mathcal{L}_0$ by adding the following syntax rules:

$$L \longrightarrow L_1 \cup L_2 \mid L_1 \circ L_2 \mid L^* \mid id(E).$$

The link constructors "$\cup$", "$\circ$", and "$*$" have their intuitive meaning of *union*, *concatenation*, and *reflexive transitive closure* of links, while $id(E)$ denotes the set of links from every object in class E to itself. It is used to test whether along a complex link a certain condition is satisfied.

To complete the constructors for complex links we consider also the *difference* of two links, which we denote with the letter $\mathcal{D}$, and which is obtained by adding the syntax rule:

$$L \longrightarrow L_1 \setminus L_2.$$

It can be used to refer to those objects that are connected via some link but not via another one. We use the following abbreviations to increase readability:

$$\begin{array}{rcl} \emptyset & := & A \setminus A, \qquad \text{for some attribute name } A^1 \\ L_1 \cap L_2 & := & L_1 \setminus (L_1 \setminus L_2). \end{array}$$

Formally, every interpretation $\mathcal{I}$ satisfies the following conditions[2]:

$$\begin{array}{rcl} (L_1 \cup L_2)^{\mathcal{I}} & = & L_1^{\mathcal{I}} \cup L_2^{\mathcal{I}} \\ (L_1 \circ L_2)^{\mathcal{I}} & = & L_1^{\mathcal{I}} \circ L_2^{\mathcal{I}} \\ (L^*)^{\mathcal{I}} & = & (L^{\mathcal{I}})^* \\ (id(E))^{\mathcal{I}} & = & \{(o,o) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid o \in E^{\mathcal{I}}\} \\ (L_1 \setminus L_2)^{\mathcal{I}} & = & L_1^{\mathcal{I}} \setminus L_2^{\mathcal{I}} \end{array}$$

The constructors for building complex links, in particular union and transitive closure of attributes, add considerably to the expressivity of $\mathcal{L}$-languages. As shown in Appendix B they allow us to establish a tight correspondence between $\mathcal{L}$-languages and Propositional Dynamic Logics [80], which are modal logics of programs developed for reasoning about program schemes. This correspondence is the basis for all decidability and complexity results developed in Chapter 5. The constructor "$\cap$", called *intersection* of links has been studied in [64, 39], but only when applied to attribute names. Its treatment is more problematic when it is used together with the other link constructors we have considered, and especially in conjunction with functionality on attributes, as shown in Section 4.4.

We introduce also two additional constructors for the definition of classes that allow to specify additional conditions on the outgoing (complex) links.

The language $\mathcal{L}\Delta$ is obtained by adding the syntax rule for the so called *repeat* constructor [151, 89]:

$$E \longrightarrow \Delta(L).$$

[1] We could have chosen to define "$\emptyset$" as an abbreviation for "$id(\bot)$", which does not refer to a specific attribute name. However, using this definition, "$\emptyset$" would not be a basic link, as defined in Section 2.1.9.

[2] In the semantics, "$\circ$" designates concatenation of binary relations, "$*$" their reflexive and transitive closure, and "$\setminus$" set difference.

Intuitively, each instance of $\Delta(L)$ is the initial object of an infinite sequence of objects, each one connected through link L to the following one. More formally, the semantics is specified by requiring that every interpretation $\mathcal{I}$ satisfies the condition:

$$(\Delta(L))^{\mathcal{I}} = \{o \in \Delta^{\mathcal{I}} \mid \exists \text{ an infinite sequence } o_1, o_2, \ldots \in \Delta^{\mathcal{I}} : o = o_1 \wedge (o_i, o_{i+1}) \in L^{\mathcal{I}}, i \geq 1\}.$$

The repeat constructor can be used in its negated form to define well-founded relations such a the part-of relation [46, 136], and inductive structures, such as lists, trees, and directed acyclic graphs, as shown later. It is worth noticing that the classes that represent these inductive structures can be treated like all other classes and can be reasoned upon using the general techniques developed in Chapter 5.

The language $\mathcal{LV}$ is obtained by adding the syntax rule for the so called *role value map* constructor [35]:

$$E \longrightarrow (L_1 \subseteq L_2).$$

We use also the following abbreviation:

$$(L_1 = L_2) := (L_1 \subseteq L_2) \sqcap (L_2 \subseteq L_1).$$

The class expression $(L_1 \subseteq L_2)$ is interpreted as the set of those objects for which all objects reached through link L_1 can also be reached through link L_2. Formally, the semantics is specified by requiring that every interpretation $\mathcal{I}$ satisfies the conditions:

$$(L_1 \subseteq L_2)^{\mathcal{I}} = \{o \in \mathcal{O}^{\mathcal{I}} \mid \{o' \mid (o, o') \in L_1^{\mathcal{I}}\} \subseteq \{o' \mid (o, o') \in L_2^{\mathcal{I}}\}\}.$$

Some examples on the use of role value maps are given in the next section. We anticipate, however, that similarly to intersection of links, role value maps need special attention in order to preserve decidability of reasoning on class expressions.

2.1.8 Structured Objects ($\mathcal{LO}$)

The languages introduced so far and their combinations allow one to specify classes with complex properties. However, they suffer from the inherent limitation that the basic elements which are classes and links are interpreted as unary and binary relations respectively. There are situations which can be modeled correctly only by means of relations of arity greater that two. One possibility is to add explicitly n-ary relationships to the language and include constructors that allow for referring to components of relationships (see [57, 144] for such an approach in the context of Description Logics). Here we take a different approach which is more in the spirit of object-oriented data models [108, 19, 97, 2], and which overcomes the limitations in expressivity imposed by unary and binary relations, without losing uniformity. We regard the interpretation domain as being constituted by a set of polymorphic objects, which can be viewed as having different structures corresponding to the different roles they can play in the modeled reality. The language contains constructors for classes that allow one to specify the structure that all instances of a class should have.

This is accomplished by the following syntax rules, where $n \geq 1$:

$$\begin{array}{lcl} E & \longrightarrow & \{E\} \mid [A_1, \ldots, A_n] \mid \langle E \mid A_1, \ldots, A_n \rangle \\ L & \longrightarrow & \ni . \end{array}$$

In order to give the formal semantics of classes and links in $\mathcal{LO}$ we have to extend the notion of interpretation. Differently from the previous languages, the domain of interpretation, which we now denote by $\mathcal{O}^{\mathcal{I}}$, is not a set of unstructured objects. Instead it is a set $\mathcal{O}^{\mathcal{I}}$ of *polymorphic objects*, that is entities having simultaneously possibly more than one structure, i.e.:

1. The structure of *individual*: an object can always be considered as having this structure, and this allows it to be referenced by other objects of the domain.

2. The structure of *tuple*: an object o having this structure can be considered as a property aggregation, which is formally defined as a partial function from a finite subset $\mathcal{A}$ of $\mathbf{AN}$ to $\mathcal{O}^{\mathcal{I}}$ with the proviso that o is uniquely determined by the set of attributes on which it is defined and by their values. In

the sequel the term tuple is used to denote an element of $\mathcal{O}^{\mathcal{I}}$ that has the structure of tuple, and we write $[A_1\colon o_1, \ldots, A_k\colon o_k]$ to denote any tuple t such that, for each $i \in \{1, \ldots, k\}$, $t[A_i]$ is defined and equal to o_i (which is called the A_i-component of t). Note that the tuple t may have other components as well, besides the A_i-components.

3. The structure of *set*: an object o having this structure can be considered as an instance aggregation, which is formally defined as a finite collection of entities in $\mathcal{O}^{\mathcal{I}}$, with the following provisos: (i) the view of o as a set is unique (except for the empty set $\{\}$), in the sense that there is at most one finite collection of entities of which o can be considered an aggregation, and (ii) no other object o' is the aggregation of the same collection. In the sequel the term set is used to denote an element of $\mathcal{O}^{\mathcal{I}}$ that has the structure of set, and we write $\{\!|o_1, \ldots, o_h|\!\}$ to denote the collection whose members are exactly $o_1, \ldots, o_h$.

Intuitively, the meaning of the constructors of $\mathcal{LO}$ can be described as follows: $\{E\}$ denotes objects that have the structure of a *set* and for which the elements of the set are instances of E. $[A_1, \ldots, A_k]$ denotes objects which have the structure of a *tuple* with at least components $A_1, \ldots, A_k$. $\langle E \mid A_1, \ldots, A_k\rangle$ specifies a set S of entities which are instances of E and for which components $A_1, \ldots, A_k$ are defined and constitute a *key* for S. This means that each element of S is uniquely determined by the set of objects to which it is connected by means of attributes $A_1, \ldots, A_k$.

Formally, the interpretation function $\cdot^{\mathcal{I}}$ is defined as follows:

- It assigns to $\ni$ a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that for each $\{\!|\ldots, o, \ldots|\!\} \in \mathcal{O}^{\mathcal{I}}$, we have that $(\{\!|\ldots, o, \ldots|\!\}, o) \in \ni^{\mathcal{I}}$.

- It assigns to every attribute A a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that, for each tuple $[\ldots, A\colon o, \ldots] \in \mathcal{O}^{\mathcal{I}}$, $([\ldots, A\colon o, \ldots], o) \in A^{\mathcal{I}}$, and there is no $o' \in \mathcal{O}^{\mathcal{I}}$ different from o such that $([\ldots, A\colon o, \ldots], o') \in A^{\mathcal{I}}$. Note that this implies that every attribute in a tuple is functional for the tuple.

- It assigns to every class expression a subset of $\mathcal{O}^{\mathcal{I}}$ such that the following conditions are satisfied:

$$\begin{array}{rcl} \{E\}^{\mathcal{I}} & = & \{\{\!|o_1, \ldots, o_h|\!\} \in \mathcal{O}^{\mathcal{I}} \mid o_1, \ldots, o_h \in E^{\mathcal{I}}\} \\ [A_1, \ldots, A_k]^{\mathcal{I}} & = & \{[A_1\colon o_1, \ldots, A_k\colon o_k] \in \mathcal{O}^{\mathcal{I}} \mid o_1, \ldots, o_k \in \mathcal{O}^{\mathcal{I}}\} \\ S = \langle E \mid A_1, \ldots, A_k\rangle^{\mathcal{I}} & \subseteq & [A_1, \ldots, A_k]^{\mathcal{I}} \cap E^{\mathcal{I}} \text{ and no distinct } o, o' \in S \\ & & \text{have the same } A_1, \ldots, A_k\text{-components.} \end{array}$$

- For the other class and link constructors it satisfies the same conditions as for the languages not containing $\mathcal{O}$, with $\mathcal{O}^{\mathcal{I}}$ replacing $\Delta^{\mathcal{I}}$.

Note that the semantics of a keyed tuple gives only a sufficient condition for an object to be an instance of the keyed tuple. In this sense declaring a key on a tuple acts as a kind of integrity constraint, and every interpretation that satisfies the constraints is accepted. This corresponds to the intuitive meaning of keys, which are regarded as constraints rather than having a definitional account. In fact, for a keyed class $\langle E \mid A_1, \ldots, A_k\rangle$, given the extension of class E and of attributes $A_1, \ldots, A_k$, there are several possible ways to choose the extension of $\langle E \mid A_1, \ldots, A_k\rangle$ in such a way that the key constraint is satisfied, and it is not clear how to select one a priori. For this reason, the kind of inferences that can be drawn from keyed tuples are limited, and their use and utility will be explained in Section 2.2, where we introduce $\mathcal{L}$-schemata.

Example 2.1.4 The class of books could for example be represented by the following class expression:

$$\texttt{Thing} \sqcap \{\texttt{BookChapter}\} \sqcap [\texttt{booktitle}, \texttt{bookauthors}, \texttt{publisher}] \sqcap$$
$$\forall \texttt{booktitle.String} \sqcap \forall \texttt{bookauthors.}\{\texttt{Person}\} \sqcap \forall \texttt{publisher.Publisher}.$$

It expresses that each book can be regarded simultaneously as an object, which in this case is also an instance of `Thing`, as a set of chapters (this may be the correct view for a student that has to study the book), and finally as a tuple containing the attributes `booktitle`, `bookauthors`, and `publisher` (this might be the correct view for a librarian). ∎

2.1.9 Achieving Maximum Expressivity ($\mathcal{LT}$, $\mathcal{LT}^-$)

In the following we use $\mathcal{LT}$ (the letter $\mathcal{T}$ stands here for "total" and not for a new constructor) as an abbreviation for the language $\mathcal{LOCQILDV}\Delta$ obtained by introducing all constructors seen so far. In fact, $\mathcal{LT}$ is a super-language of the language introduced in [58] and borrows from it the basic ideas.

We will see in Section 4.4 that the expressivity of $\mathcal{LT}$ makes reasoning about class expressions (and therefore also about schemata as defined in Section 2.2) problematic. In particular, it is already undecidable whether for a generic class expression E of $\mathcal{LT}$ there is an interpretation $\mathcal{I}$ in which $E^{\mathcal{I}}$ is nonempty. Therefore we define the language $\mathcal{LT}^-$ in which syntactic restrictions are posed on the combination of certain constructors. More precisely, the use of number restrictions, difference of links, and role value maps has to be restricted. In $\mathcal{LT}^-$ we distinguish between *basic links*, denoted with B, and arbitrarily complex links. The syntax is specified by the following rules:

$$\begin{array}{lcl} E & \longrightarrow & C \mid \{E\} \mid [A_1, \ldots, A_n] \mid \langle E \mid A_1, \ldots, A_n \rangle \mid \\ & & E_1 \sqcap E_2 \mid \neg E \mid \forall L.E \mid \exists^{\leq n} B.E \mid \exists^{\leq n} B^-.E \mid \\ & & (B_1 \subseteq B_2) \mid (B_1^- \subseteq B_2^-) \mid \Delta(L) \\ B & \longrightarrow & A \mid \ni \mid B_1 \cup B_2 \mid B_1 \setminus B_2 \\ L & \longrightarrow & B \mid L_1 \cup L_2 \mid L_1 \circ L_2 \mid L^- \mid L^* \mid id(E). \end{array}$$

We have chosen here to introduce only a minimal set of constructors that achieve the desired expressivity, and we adopt all abbreviations introduced so far to increase readability: Since general negation is available, disjunction can be expressed by means of conjunction and qualified existential quantification by means of universal quantification. $\top$ and $\bot$ are regarded as abbreviations for $C \sqcup \neg C$ and $C \sqcap \neg C$, respectively, where C is an arbitrary class name. Also, number restrictions of the form $\exists^{\geq m} L.E$, with m a positive integer, are expressed as $\neg\exists^{\leq m-1} L.E$.

The semantics for $\mathcal{LT}^-$ class and link expressions is defined exactly as for $\mathcal{LT}$, considering that basic links are just particular types of complex links.

One of the objectives of this thesis is to show that reasoning in $\mathcal{LT}^-$ is decidable. We will see in Section 5.2 that the careful choice of constructors in $\mathcal{LT}^-$ is sufficient to avoid all problems encountered in $\mathcal{LT}$, and that reasoning in $\mathcal{LT}^-$ can be performed in deterministic exponential time. We argue also that in a certain sense $\mathcal{LT}^-$ offers the maximum expressivity that can be achieved without losing decidability of reasoning, since all limitations we have imposed with respect to $\mathcal{LT}$ are indeed necessary to preserve decidability.

2.1.10 Summary

Tables 2.1 and 2.2 summarize syntax and semantics of all constructors on classes and attributes, respectively, that are used in this thesis. In both tables we have separated those constructors that are applicable only when considering a structured domain. Figure 2.1 shows a taxonomy of (most of the) $\mathcal{L}$-languages that we consider in this thesis. For each language we have included also a reference to the sections or chapters where it is used or its properties are studied. The two languages $\mathcal{LUF}_{\cap,\circ}$ and $\mathcal{LUF}(\circ,\cup)$ are those referred to in Theorems 4.4.2 and 4.4.3, respectively. $\mathcal{LUF}_{\cap,\circ}$ is the language obtained from $\mathcal{LUF}$ by adding also the possiblity to use intersection and concatenation of attributes inside universal quantification (but not inside functional restrictions). $\mathcal{LUF}(\circ,\cup)$ is obtained from $\mathcal{LUF}$ by adding the possiblity to express functionality of link expressions constructed using concatenation and union of attributes. Since these languages do not appear anywhere else in the thesis, they are named explicitly only in Figure 2.1.

$\mathcal{L}$-languages are an object of interest on their own. In the last decade researchers have studied intensively the problem of reasoning on class expressions of the various languages. It has turned out that the basic problem to be studied in this context is that of *subsumption*, which is the task of deciding, given two class expression E_1 and E_2, if the extension of E_1 is a subset of the extension of E_2 for all possible interpretations. The other typical forms of reasoning that can be performed on class expressions, such as checking whether an expression is consistent or whether two expressions are equivalent, can easily be reduced to subsumption. The focus of research has been on analyzing the tradeoff between complexity of subsumption between class expressions and expressivity of the underlying language. The research was initiated by the seminal paper by Brachman and Levesque [32], which pointed out how even a seemingly small change in expressivity possibly results in a big change in complexity of subsumption. Subsequent research was intended to exactly

Constructor Name		Syntax	Semantics
class name		C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
top		$\top$	$\Delta^{\mathcal{I}}$
bottom		$\bot$	$\emptyset$
atomic negation		$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction		$E_1 \sqcap E_2$	$E_1^{\mathcal{I}} \cap E_2^{\mathcal{I}}$
universal quantif.		$\forall L.E$	$\{o \mid \forall o' \colon (o,o') \in L^{\mathcal{I}} \rightarrow o' \in E^{\mathcal{I}}\}$
existential quantif.		$\exists L$	$\{o \mid \exists o' \colon (o,o') \in L^{\mathcal{I}}\}$
qualif. exist. quantif.	$\mathcal{E}$	$\exists L.E$	$\{o \mid \exists o' \colon (o,o') \in L^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}}\}$
disjunction	$\mathcal{U}$	$E_1 \sqcup E_2$	$E_1^{\mathcal{I}} \cup E_2^{\mathcal{I}}$
general negation	$\mathcal{C}$	$\neg E$	$\Delta^{\mathcal{I}} \setminus E^{\mathcal{I}}$
number restrictions	$\mathcal{N}$	$\exists^{\geq m} L$	$\{o \mid \sharp\{o' \mid (o,o') \in L^{\mathcal{I}}\} \geq m\}$
		$\exists^{\leq n} L$	$\{o \mid \sharp\{o' \mid (o,o') \in L^{\mathcal{I}}\} \leq n\}$
qualified number restrictions	$\mathcal{Q}$	$\exists^{\geq m} L.E$	$\{o \mid \sharp\{o' \mid (o,o') \in L^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}}\} \geq m\}$
		$\exists^{\leq n} L.E$	$\{o \mid \sharp\{o' \mid (o,o') \in L^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}}\} \leq n\}$
repeat	Δ	$\Delta(L)$	$\{o_0 \mid \exists o_1, o_2, \ldots \colon (o_i, o_{i+1}) \in L^{\mathcal{I}}, i \geq 0\}$
role value map	$\mathcal{V}$	$(L_1 \subseteq L_2)$	$\{o \mid \{o' \mid (o,o') \in L_1^{\mathcal{I}}\} \subseteq \{o' \mid (o,o') \in L_2^{\mathcal{I}}\}\}$
set	$\mathcal{O}$	$\{E\}$	$\{\{\!\!\{o_1, \ldots, o_h\}\!\!\} \in \mathcal{O}^{\mathcal{I}} \mid o_1, \ldots, o_h \in E^{\mathcal{I}}\}$
tuple	$\mathcal{O}$	$[A_1, \ldots, A_k]$	$\{[A_1 \colon o_1, \ldots, A_k \colon o_k] \in \mathcal{O}^{\mathcal{I}} \mid o_1, \ldots, o_k \in \mathcal{O}^{\mathcal{I}}\}$
tuple with key	$\mathcal{O}$	$\langle E \mid A_1, \ldots, A_k \rangle$	$\subseteq [A_1, \ldots, A_k]^{\mathcal{I}} \cap E^{\mathcal{I}}$ and key condition

Table 2.1: Syntax and semantics of the class forming constructors

Constructor Name		Syntax	Semantics
attribute name		A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
inverse	$\mathcal{I}$	L^-	$\{(o,o') \mid (o',o) \in L^{\mathcal{I}}\}$
union	$\mathcal{L}$	$L_1 \cup L_2$	$L_1^{\mathcal{I}} \cup L_2^{\mathcal{I}}$
concatenation	$\mathcal{L}$	$L_1 \circ L_2$	$L_1^{\mathcal{I}} \circ L_2^{\mathcal{I}}$
transitive closure	$\mathcal{L}$	L^*	$(L^{\mathcal{I}})^*$
identity	$\mathcal{L}$	$id(E)$	$\{(o,o) \mid o \in E^{\mathcal{I}}\}$
difference	$\mathcal{D}$	$L_1 \setminus L_2$	$L_1^{\mathcal{I}} \setminus L_2^{\mathcal{I}}$
intersection	$\mathcal{D}$	$L_1 \cap L_2$	$L_1^{\mathcal{I}} \cap L_2^{\mathcal{I}}$
empty link	$\mathcal{D}$	$\emptyset$	$\emptyset$
member	$\mathcal{O}$	$\ni$	$\{(\{\!\!\{\ldots, o, \ldots\}\!\!\}, o) \in \mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}\}$

Table 2.2: Syntax and semantics of the link forming constructors

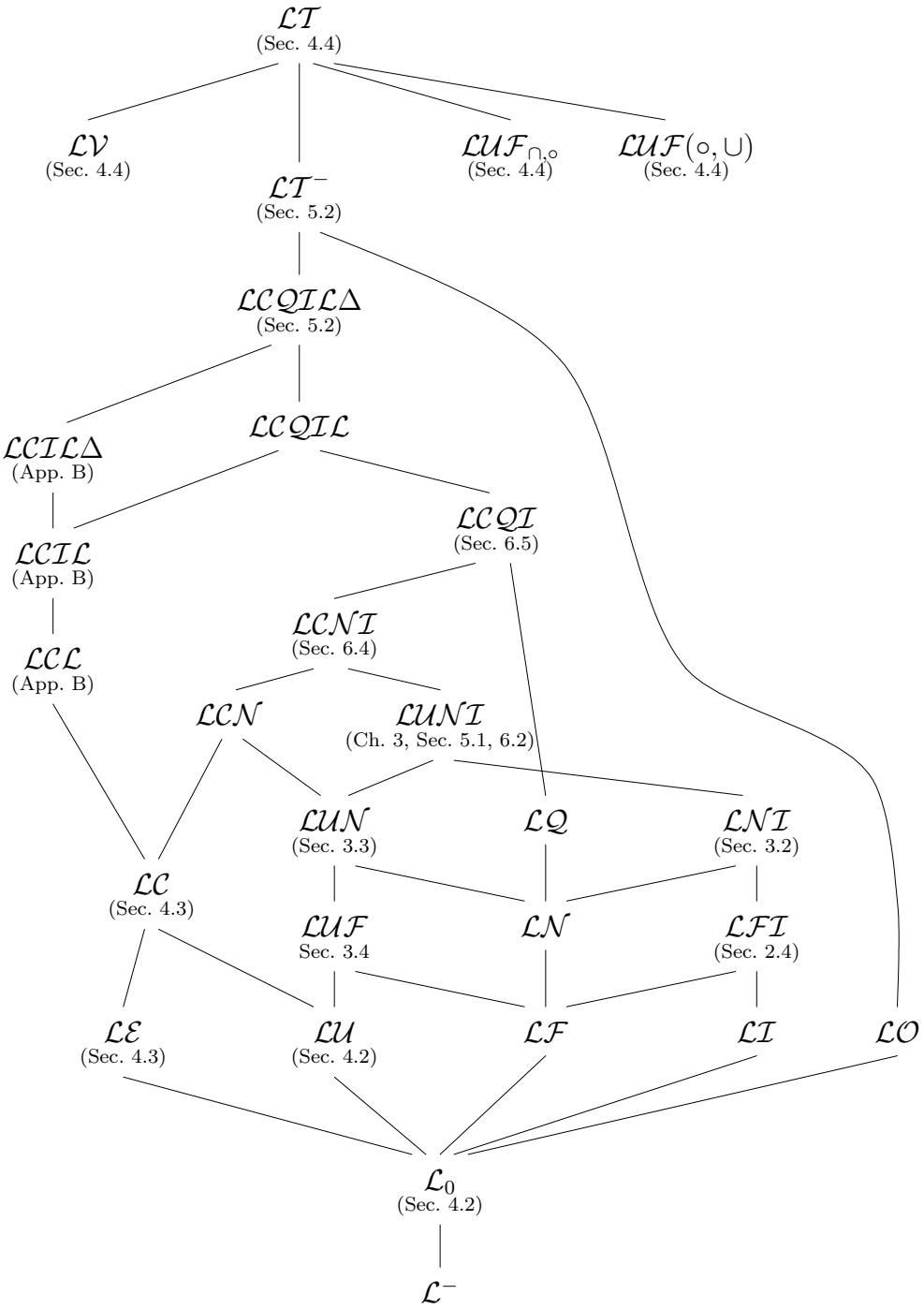

Figure 2.1: The lattice of $\mathcal{L}$-languages

characterize the contributions that each constructor and the interaction of different constructors give to the complexity of reasoning [120, 142, 95, 143, 63, 64, 62], and we now have a reasonably good and complete understanding of the whole field [64, 65].

2.2 $\mathcal{L}$-Schemata

Using class expressions of an $\mathcal{L}$-language, intensional knowledge can be specified through $\mathcal{L}$-schemata. We follow the terminology introduced in [138] in the context of Description Logics.

Definition 2.2.1 Given a language $\mathcal{L}$, an *$\mathcal{L}$-schema* is a triple $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$, where

- $\mathcal{C} \subseteq \mathbf{CN}$ is a finite set of class names.
- $\mathcal{A} \subseteq \mathbf{AN}$ is a finite set of attribute names.
- $\mathcal{T}$ is a finite set of *assertions* in $\mathcal{L}$. Each such assertion has one of the forms

$$\begin{array}{lcll} C & \dot{\preceq} & E & \textit{(primitive class specification)} \\ C & \doteq & E & \textit{(class definition)}, \end{array}$$

  where $C \in \mathcal{C}$ and E is a class expression of $\mathcal{L}$ involving only names of $\mathcal{C} \cup \mathcal{A}$. ∎

Primitive class specifications are used to specify necessary conditions for an object to be an instance of a class, while class definitions specify both necessary and sufficient conditions. We call *introduction of* C an assertion in which the class name C appears in the left-hand side.

When no confusion can arise, we may abuse terminology, by saying that a schema $(\mathcal{C}, \mathcal{A}, \mathcal{T})$ coincides with its set $\mathcal{T}$ of assertions. In such a case we assume implicitly that the set $\mathcal{N}$ of names is constituted by exactly those names that appear in the assertions in $\mathcal{T}$. Also, when not specified otherwise we assume that $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$.

Both in Knowledge Representation and in database models several different assumptions on the form of a schema are made, either implicitly or explicitly.

A common assumption is that the following condition is satisfied (see for example [121, 9, 108, 25]):

I. Each class name has at most one introduction.

Under this assumption, a class name appearing in a schema on the left-hand side of a class definition is called a *defined class*, a class name appearing on the left-hand side of a primitive class specification is called a *primitive class*, and a class name not appearing on the left-hand side of any assertion is called an *atomic class*.

2.2.1 Cycles in $\mathcal{L}$-Schemata

We formally define a cycle in a schema $\mathcal{S}$ as follows [122]: A class name C *directly uses* a class name C' if and only if C' appears in the right hand side of an introduction of C in $\mathcal{S}$. A class name C_0 *uses* C_n if there is a chain $C_0, C_1, \ldots, C_n$ such that C_i directly uses C_{i+1}, $i \in \{1, \ldots, n-1\}$. Finally, we say that $\mathcal{S}$ contains a cycle if some class name uses itself.

We say that a schema is *acyclic* if it satisfies both conditions I above and II below:

II. The schema contains no *cycles*.

For acyclic schemata the semantics is specified as follows: An interpretation $\mathcal{I}$ *satisfies* an assertion $C \dot{\preceq} E$ if and only if $C^{\mathcal{I}} \subseteq E^{\mathcal{I}}$, and it satisfies an assertion $C \doteq E$ if and only if $C^{\mathcal{I}} = E^{\mathcal{I}}$. An interpretation that satisfies all assertions in a schema $\mathcal{S}$ is called a *model* of $\mathcal{S}$.

In most existing concept-based knowledge representation systems, the usual assumption is that the schema defining the intensional part of a knowledge base is acyclic [121]. Imposing this condition, however, strongly limits the expressive power of the system, since many real world concepts can be expressed naturally only in a recursive way and thus through cycles. Take as an example the class `Person`, which can be defined by the assertion

$$\texttt{Person} \doteq \texttt{Mammal} \sqcap \exists^{=2}\texttt{parent} \sqcap \forall \texttt{parent.Person}.$$

Schemata possibly containing cycles (but still satisfying condition I) are called *general*. General schemata have been studied intensively by researchers working in Knowledge Representation [122, 9, 8, 141, 60].

Different semantics for the interpretation of cycles have been proposed. The one obtained by the same specification as above for acyclic schemata is called *descriptive semantics*. It interprets each assertion in a schema as a constraint that the interpretation must satisfy, and each interpretation satisfying all constraints imposed by the assertions is a model of the schema. By considering each class introduction as an equation, the descriptive semantics is the one that accepts as a model any interpretation that is a fixpoint of the equations.

Fixpoint semantics are obtained by regarding only particular fixpoints of the equations as models. Two kinds of fixpoint semantics have been considered: the *least* and *greatest fixpoint semantics*, obtained by allowing only interpretations that are respectively the least and greatest fixpoint. There has been an intense debate among researchers in knowledge representation on which is the most appropriate semantics to be adopted [122, 9], and there have also been recent proposals for an integration of the different types of semantics [55, 141].

A more detailed discussion on the different types of semantics is outside the scope of this thesis, and in the following we assume to adopt the descriptive semantics. This semantics has been advocated to be the most natural one (see for example [37, 39]) and it is also the only one that can be extended in a natural way to free schemata defined below. Notice again that in the absence of cycles this problem does not arise at all since all three semantics coincide.

2.2.2 Primitive $\mathcal{L}$-Schemata

Another restriction which is sometimes made in concept-based knowledge representation systems (see for example [36]) and which is usually implicit in database models, is the following:

III. The schema contains only primitive class specifications and no class definitions.

Schemata satisfying condition III are called *primitive*. For such types of schemata, condition I may be withdrawn without influencing expressivity (assuming that the language contains the constructor for expressing conjunction of class expressions, which is the case for all languages studied in this thesis). In fact, if the schema contains two assertions $A \dot{\preceq} E_1$ and $A \dot{\preceq} E_2$, we may substitute them with the equivalent assertion $A \dot{\preceq} E_1 \sqcap E_2$.

2.2.3 Free $\mathcal{L}$-Schemata

If none of the above three conditions is required to hold, we obtain the so called *free* schemata. For free schemata the distinction between defined, primitive, and atomic classes makes no sense. It is easy to see that the expressivity of free schemata (that allow also for class definitions) is equivalent to the one of schemata constituted by *free inclusion assertions*, which have the form

$$E_1 \ \dot{\preceq} \ E_2,$$

where both E_1 and E_2 are arbitrary class expressions with no restriction at all. In fact, an inclusion assertion of the form $E_1 \dot{\preceq} E_2$ can be simulated by introducing an additional class name C and using the pair of assertions

$$\begin{aligned} C &\doteq E_1 \\ C &\dot{\preceq} E_2. \end{aligned}$$

Conversely, a class definition $C \doteq E$ is equivalent to the pair of free assertions

$$\begin{aligned} C &\dot{\preceq} E \\ E &\dot{\preceq} C. \end{aligned}$$

2.2.4 Summary and Examples

Table 2.3 summarizes the different types of assertions that we have considered, and Table 2.4 shows the conditions satisfied by the various types of $\mathcal{L}$-schemata.

Assertion	Syntax	Semantics
primitive class specification	$C \dot{\preceq} E$	$C^{\mathcal{I}} \subseteq E^{\mathcal{I}}$
class definition	$C \doteq E$	$C^{\mathcal{I}} = E^{\mathcal{I}}$
free assertion	$E_1 \dot{\preceq} E_2$	$E_1^{\mathcal{I}} \subseteq E_2^{\mathcal{I}}$

Table 2.3: Syntax and semantics of the assertions in schemata

Assertion	acyclic	cycles allowed
primitive specifications	acyclic primitive (II, III)	general primitive (III)
class definitions	acyclic (I, II)	general (I)
free assertions	free (no condition)	
(I) each class has at most one introduction (II) no cycles (III) no class definitions		

Table 2.4: Conditions satisfied by the different types of schemata

We give now a few examples to illustrate the expressivity of $\mathcal{L}$-schemata, using the different constructors that were introduced in Section 2.1. In particular, all schemata defined below use only constructors of $\mathcal{LT}^-$.

Example 2.2.2 The following $\mathcal{L}$-schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$, which models some facts about a condominium, gives an example of the use of polymorphism:

$\mathcal{C}$:= {Apartment, Condominium, Manager, CondMan, Address, String, Integer},

$\mathcal{A}$:= {location, budget, manages, ssn, city, street, num},

and the set $\mathcal{T}$ of assertions consists of:

$$
\begin{array}{rcl}
\texttt{Apartment} & \dot{\preceq} & \cdots \\
\texttt{Condominium} & \dot{\preceq} & \{\texttt{Apartment}\} \sqcap [\texttt{location}, \texttt{budget}] \sqcap \\
 & & \forall \texttt{location.Address} \sqcap \forall \texttt{budget.Integer} \sqcap \\
 & & \langle \texttt{Condominium} \mid \texttt{location} \rangle \sqcap \\
 & & \forall \texttt{manages}^-\texttt{.Manager} \sqcap \exists^{=1}\texttt{manages}^- \\
\texttt{Manager} & \dot{\preceq} & [\texttt{ssn}, \texttt{location}] \sqcap \forall \texttt{ssn.String} \sqcap \forall \texttt{location.Address} \sqcap \\
 & & [\texttt{Manager}]\texttt{ssn} \sqcap \\
 & & \exists \texttt{manages} \\
\texttt{CondMan} & \doteq & \texttt{Manager} \sqcap \exists \texttt{manages.Condominium} \\
\texttt{Address} & \dot{\preceq} & \langle \texttt{Address} \mid \texttt{city}, \texttt{street}, \texttt{num} \rangle \sqcap \\
 & & \forall \texttt{city} \cup \texttt{street.String} \sqcap \forall \texttt{num.Integer}
\end{array}
$$

The class Condominium is specified using a conjunction of the set structure {Apartment} and the record structure [location, budget]. Therefore, the designer of such a schema is anticipating that each instance of Condominium will be used both as a set (in this case the set of apartments forming the condominium) and as a record structure collecting the relevant attributes of the condominium (in this case where the condominium is located and its budget). The location is a key for the condominium, implying that there cannot be two different condominia located at the same address. Moreover, each instance of condominium can also be regarded as an individual that can be referred to by other objects through links (in this case manages). ∎

We illustrate how $\mathcal{L}$-schemata can be used to define inductive structures on the examples of lists and graphs. We argue that the ability to define inductive structures in an $\mathcal{L}$-schema is an important enhancement with respect to traditional data models, where such structures, if present at all, are ad hoc additions requiring a special treatment by procedures that perform inferences on a schema [43, 18].

Example 2.2.3 Typically, the class of lists is defined inductively as the smallest set *List* such that:

- *Nil* is a *List*, and

- every pair whose first element is any object, and whose second element is a *List*, is a *List*.

This inductive definition is captured by the class `List` in the $\mathcal{L}$-schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$, where

$\mathcal{C} := \{\texttt{Nil}, \texttt{List}\}$,
$\mathcal{A} := \{\texttt{first}, \texttt{rest}\}$,

and the set $\mathcal{T}$ of assertions consists of:

$$\begin{aligned} \texttt{List} &\doteq \texttt{Nil} \sqcup ([\texttt{first}, \texttt{rest}] \sqcap \forall \texttt{rest}.\texttt{List}) \sqcap \neg\Delta(\texttt{rest}) \sqcap \\ &\quad\ \exists^{\leq 1}\texttt{rest}^- \\ \texttt{Nil} &\doteq \forall \texttt{first} \sqcup \texttt{rest}.\bot. \end{aligned}$$

Notice that `List` appears on the left-hand side of a class definition, and is defined recursively, in the sense that the term `List` we are defining occurs in the body of the definition. In general, a recursive definition should not be confused with an inductive one: an inductive definition selects the smallest set satisfying a certain condition, while a recursive one simply states the condition without specifying any selection criteria to choose among all possible sets satisfying it. In fact, the negated repeat constructor accomplishes this selection, making our recursive definition of `List` inductive. Observe also the use of a number restriction which forbids that two lists share a common sublist.

Once lists are defined they can be easily specialized by selecting for example the kind of information contained in an element or additional structural constraints, such as a specific length:

$$\begin{aligned} \texttt{ListOfPers} &\doteq \texttt{List} \sqcap \forall \texttt{rest}^* \circ \texttt{first}.\texttt{Person} \\ \texttt{ListOf2Pers} &\doteq \texttt{ListOfPers} \sqcap \exists \texttt{rest} \circ \texttt{rest} \sqcap \forall \texttt{rest} \circ \texttt{rest} \circ \texttt{rest}.\bot. \end{aligned}$$

■

Example 2.2.4 The following $\mathcal{L}$-schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ models several variants of rooted graphs, including finite directed acyclic graphs (`DAG`) and finite trees (`Tree`).

$\mathcal{C} := \{\texttt{Graph}, \texttt{DAG}, \texttt{Tree}, \texttt{BinaryGraph}, \texttt{BinaryTree}\}$,
$\mathcal{A} := \{\texttt{label}, \texttt{edge}, \texttt{left}, \texttt{right}\}$,

and the set $\mathcal{T}$ of assertions consists of:

$$\begin{aligned} \texttt{Graph} &\doteq \exists^{=1}\texttt{label} \sqcap \forall \texttt{edge}.\texttt{Graph} \\ \texttt{DAG} &\doteq \texttt{Graph} \sqcap \neg\Delta(\texttt{edge}) \\ \texttt{Tree} &\doteq \texttt{Graph} \sqcap \forall \texttt{edge}.\texttt{Tree} \sqcap \neg\Delta(\texttt{edge}) \sqcap \exists^{\leq 1}\texttt{edge}^- \\ \texttt{BinaryGraph} &\doteq \texttt{Graph} \sqcap \forall \texttt{edge}.\texttt{BinaryGraph} \sqcap \exists^{\leq 2}\texttt{edge} \\ \texttt{BinaryTree} &\doteq \texttt{Graph} \sqcap \forall \texttt{edge}.\texttt{BinaryTree} \sqcap \neg\Delta(\texttt{edge}) \sqcap \\ &\quad\ \exists^{\leq 1}\texttt{edge}^- \sqcap \exists^{\leq 1}\texttt{left} \sqcap \exists^{\leq 1}\texttt{right} \sqcap \\ &\quad\ (\texttt{left} \sqcup \texttt{right} = \texttt{edge}) \sqcap (\texttt{left} \subseteq \texttt{edge} \setminus \texttt{right}) \end{aligned}$$

Each graph is identified by its root, which has a label (that can be an arbitrary object) and is connected through attribute `edge` only to instances of `Graph`. A `DAG` is a graph for which the edge relation is well-founded, which is expressed as in Example 2.2.3 by means of a negated repeat constructor. A `Tree` satisfies the additional constraint that each node has at most one incoming edge, expressed through a functional restriction on the inverse of the attribute `edge`. A `BinaryGraph` is a `Graph` for which each node has at most two outgoing edges. Finally, a `BinaryTree` has at most one left and one right successor, which are distinct instances of `BinaryTree`. The left and right successors are exactly the objects that can be reached through the attribute `edge`, which is well-founded, and whose inverse is functional. ■

2.3 Schema Level Reasoning

As we already mentioned in Chapter 1, several forms of reasoning at the schema level may be considered of interest. We now formally define the most important ones , which are those dealt with in this work.

2.3.1 Reasoning Services

Definition 2.3.1 Given an $\mathcal{L}$-language $\mathcal{L}$, an $\mathcal{L}$-schema $\mathcal{S}$ and two class expressions E, E' of $\mathcal{L}$, we call[3]:

- *Schema consistency* the problem of establishing whether $\mathcal{S}$ is *consistent* (written as $\mathcal{S} \not\models_u$), i.e. whether it admits a model.
- *Class consistency* the problem of establishing whether E is *consistent in* $\mathcal{S}$ (written as $\mathcal{S} \not\models_u E \equiv \bot$), i.e. whether there is a model $\mathcal{I}$ of $\mathcal{S}$ such that $E^{\mathcal{I}} \neq \emptyset$.
- *Class subsumption* the problem of establishing whether E is *subsumed by* E' *in* $\mathcal{S}$ (written as $\mathcal{S} \models_u E \preceq E'$), i.e. whether $E^{\mathcal{I}} \subseteq E'^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{S}$. E is called the *subsumee*, and E' is called the *subsumer*.
- *Class equivalence* the problem of establishing whether E is *equivalent to* E' *in* $\mathcal{S}$ (written as $\mathcal{S} \models_u E \equiv E'$), i.e. whether $E^{\mathcal{I}} = E'^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{S}$. ∎

The negated forms of these problems are denoted in the obvious way by replacing $\models_u$ with $\not\models_u$ or vice versa. For the latter three problems, we may also consider restricted forms obtained by requiring that one or both of the classes for which we have to check consistency, subsumption, or equivalence are class names or some restricted form of class expression.

Example 2.2.3 (cont.) We define now *NestedList* as the smallest set such that:

- *Nil* is a *NestedList*, and
- every pair whose first element is either an *Atom* or a *NestedList*, and whose second element is a *NestedList*, is a *NestedList*.

Such structure is captured by the following set $\mathcal{T}''$ of assertions:

$$\begin{aligned} \texttt{NestedList} &\doteq \texttt{Nil} \sqcup ([\texttt{first}, \texttt{rest}] \sqcap \neg\Delta(\texttt{first} \sqcup \texttt{rest}) \sqcap \\ &\quad \forall \texttt{first}.(\texttt{Atom} \sqcup \texttt{NestedList}) \sqcap \forall \texttt{rest}.\texttt{NestedList}) \sqcap \\ &\quad \exists^{\leq 1}\texttt{rest}^{-} \\ \texttt{Atom} &\dot{\preceq} \neg\texttt{Nil} \sqcap \forall \texttt{first} \sqcup \texttt{rest}.\bot. \end{aligned}$$

Let now $\mathcal{S}' := (\mathcal{C}', \mathcal{A}, \mathcal{T}')$, where $\mathcal{C}' := \mathcal{C} \cup \{\texttt{Atom}, \texttt{NestedList}\}$ and $\mathcal{T}' := \mathcal{T} \cup \mathcal{T}''$. Then it is possible to show that

$$\begin{aligned} \mathcal{S}' &\models_u \texttt{Atom} \sqcap \texttt{List} \equiv \bot \\ \mathcal{S}' &\models_u \texttt{NestedList} \preceq \texttt{List}. \end{aligned}$$

While the first inference is rather trivial, the second one requires to take into account that the definitions of `List` and `NestedList` are inductive. It would not hold any more, if we had just used recursive assertions without making use of the negated repeat constructor. ∎

Example 2.2.4 (cont.) By using class subsumption we can compute the lattice of graphs shown in Figure 2.2, where an arrow from class C to C' means that $\mathcal{S} \models_u C \preceq C'$. Notice again, that while most of the inferences are trivial, in order to deduce that $\mathcal{S} \models_u \texttt{BinaryTree} \preceq \texttt{BinaryGraph}$ the inductive nature of the definitions must be exploited. Moreover, to infer that every instance of `BinaryTree` satisfies $\exists^{\leq 2}\texttt{edge}$, one needs to reason on the number restrictions and role value maps together. ∎

[3]The subscript "u" in the symbol $\models$ stands for "unrestricted", to distinguish this case from the finite one which is defined later.

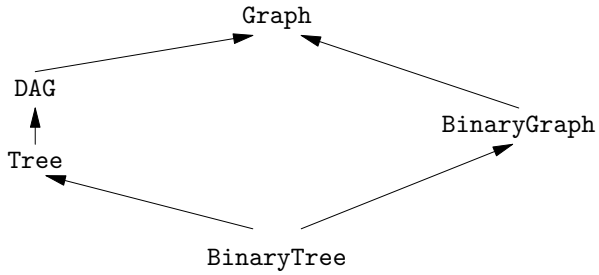

Figure 2.2: A lattice of graphs

These reasoning services in connection with an expressive formalism for the definition of schemata can be profitably exploited in several contexts. Indeed, a rich formalism permits capturing more semantics at the schema level, thus allowing the designer to declaratively represent relevant knowledge about the classes of the application. It follows that sophisticated types of constraints can be asserted in the schema, rather than being embedded in procedures that manipulate the database, with the advantage of devising general integrity checking methods to be included in future database systems. In addition, expressing more knowledge at the schema level implies more possibilities to reason about the intension of the database:

- Automated methods for checking schema and class consistency can support the designer of the database, allowing her to eliminate inconsistencies whose detection may involve complicated inferences. Classes that are equivalent constitute a redundancy that should not be present in the schema, and such a task can be performed via class subsumption and class equivalence.
- In using the database, for example in type checking, a compiler may exploit class subsumption to decide if method invocations are well-typed.
- Schema level reasoning may be used in the process of query answering for performing semantic query optimization [25, 40] or for providing intensional answers [24].
- New problems posed by cooperative and distributed information systems, such as schema comparison and integration [42], may take advantage of all the reasoning services described above.

The reasoning services introduced in Definition 2.3.1 are not independent from each other, since the following relations hold (We state them here in form of a proposition for future reference).

Proposition 2.3.2 *Let $\mathcal{L}$ be an $\mathcal{L}$-language, $\mathcal{S}$ be an $\mathcal{L}$-schema, and E, E' two $\mathcal{L}$-class expressions. Then the following holds:*

1. $(\mathcal{S} \not\models_u) \iff (\mathcal{S} \not\models_u \top \equiv \bot)$.
2. $(\mathcal{S} \models_u E \preceq E') \iff (\mathcal{S} \models_u E \sqcap \neg E' \equiv \bot)$.
3. $(\mathcal{S} \models_u E \equiv E') \iff (\mathcal{S} \models_u E \preceq E' \wedge \mathcal{S} \models_u E' \preceq E)$.

Proof. Straightforward from the semantics of $\mathcal{L}$-languages and $\mathcal{L}$-schemata. □

Proposition 2.3.2 shows that schema consistency can always be reduced to class consistency, and that in those languages closed under negation also subsumption and equivalence can be reduced to class consistency. This leads us to regard class consistency as the main reasoning task when performing intensional reasoning on a schema.

Additionally, depending on the form of the schema, and on the allowed constructors some of the problems we have considered may be trivial to decide. Two examples of this are given by the following easy propositions.

Proposition 2.3.3 *For every $\mathcal{L}^-$-schema $\mathcal{S}$, every class expression is consistent in $\mathcal{S}$.*

Proof. As in the proof of Proposition 2.1.2 we define an interpretation $\mathcal{I} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}} := \{o\}$, $C^{\mathcal{I}} := \{o\}$ for any class name $C \in \mathcal{C}$, and $A^{\mathcal{I}} := \{(o, o)\}$ for any attribute name $A \in \mathcal{A}$. Then for any $\mathcal{L}^-$-class expression E constructed using class names in $\mathcal{C}$ and attribute names in $\mathcal{A}$ we have that $E^{\mathcal{I}} = \{o\}$. Therefore $\mathcal{I}$ is a model of $\mathcal{S}$ in which every class expression has a nonempty extension. □

Proposition 2.3.4 *Every primitive $\mathcal{L}$-schema is consistent.*

Proof. Consider a primitive $\mathcal{L}$-schema $\mathcal{S}$ and any interpretation $\mathcal{I}$ that assigns to every class name the empty set as extension. Then $\mathcal{I}$ satisfies any primitive class specification in $\mathcal{S}$, independently of the class expression on the right-hand side, since the class name on the left hand side is interpreted as the empty set. Therefore $\mathcal{I}$ is a model of $\mathcal{S}$. □

The next proposition shows that without loss of generality it is sufficient to consider restricted forms of class consistency and class subsumption.

Proposition 2.3.5 *Let $\mathcal{S}$ be a primitive $\mathcal{L}$-schema, E_1, E_2 be two class expressions using only names in $\mathcal{S}$, $C \notin \mathcal{C}$ be an additional class name, and $\mathcal{S}' := (\mathcal{C}', \mathcal{A}, \mathcal{T}')$, with $\mathcal{C}' := \mathcal{C} \cup \{C\}$ and $\mathcal{T}' := \mathcal{T} \cup \{C \doteq E_1\}$. Then the following holds:*

1. $\mathcal{S} \not\models_u E_1 \equiv \bot \iff \mathcal{S}' \not\models_u C \equiv \bot$.
2. $\mathcal{S} \models_u E_1 \preceq E_2 \iff \mathcal{S}' \models_u C \preceq E_2$.

Proof. (1) "⇐" Let $\mathcal{I}$ be a model of $\mathcal{S}$ such that $E_1^{\mathcal{I}} \neq \emptyset$. We extend $\mathcal{I}$ to an interpretation of $\mathcal{S}'$ by setting $C^{\mathcal{I}} := E_1^{\mathcal{I}}$. Since C appears in no assertion of $\mathcal{T}$, and since $\mathcal{I}$ satisfies also the additional assertion in $\mathcal{T}'$, $\mathcal{I}$ is also a model of $\mathcal{S}'$ in which $C^{\mathcal{I}} \neq \emptyset$. "⇒" Let $\mathcal{I}$ be a model of $\mathcal{S}'$ such that $C^{\mathcal{I}} \neq \emptyset$. Then $\mathcal{I}$ is also a model of $\mathcal{S}$, and since $E_1^{\mathcal{I}} \supseteq C^{\mathcal{I}}$ and $C^{\mathcal{I}} \neq \emptyset$, we have also that $E_1^{\mathcal{I}} \neq \emptyset$.

(2) "⇐" Let $\mathcal{I}$ be a model of $\mathcal{S}$ such that $E_1^{\mathcal{I}} \not\subseteq E_2^{\mathcal{I}}$. We extend $\mathcal{I}$ to an interpretation of $\mathcal{S}'$ by setting $C^{\mathcal{I}} := E_1^{\mathcal{I}}$. Then $\mathcal{I}$ is also a model of $\mathcal{S}'$ in which $C^{\mathcal{I}} \not\subseteq E_2^{\mathcal{I}}$. "⇒" Let $\mathcal{I}$ be a model of $\mathcal{S}'$ such that $C^{\mathcal{I}} \not\subseteq E_2^{\mathcal{I}}$. Then $\mathcal{I}$ is also a model of $\mathcal{S}$, and since $E_1^{\mathcal{I}} \supseteq C^{\mathcal{I}}$ and $C^{\mathcal{I}} \not\subseteq E_2^{\mathcal{I}}$, we have also that $E_1^{\mathcal{I}} \not\subseteq E_2^{\mathcal{I}}$. □

Part 1 of Proposition 2.3.5 shows that when verifying class consistency we can restrict our attention to the case where the class expression to be verified for consistency is constituted just by a single class name. Similarly, by part 2 it is sufficient to consider the restricted form of class subsumption where the subsumed class expression is just a class name.

2.3.2 Equivalence of Schemata

The reasoning procedures presented in the following chapters will be applicable only to schemata that are in special forms. In order to show that this is indeed no limitation, we first formally define *equivalence of schemata* and then show that the result of applying the reasoning tasks are invariant with respect to equivalent schemata.

Definition 2.3.6 Two schemata $\mathcal{S} = (\mathcal{C}, \mathcal{A}, \mathcal{T})$ and $\mathcal{S}' = (\mathcal{C}', \mathcal{A}', \mathcal{T}')$, where $\mathcal{C} \subseteq \mathcal{C}'$ and $\mathcal{A} \subseteq \mathcal{A}'$ are said to be *equivalent* if the following holds:

- Every model $\mathcal{I} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{S}$ can be extended to a model $\mathcal{I}' := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}'})$ of $\mathcal{S}'$ by interpreting the class names in $\mathcal{C}' \setminus \mathcal{C}$ and the attribute names in $\mathcal{A}' \setminus \mathcal{A}$ in an appropriate way.
- Every model $\mathcal{I}'$ of $\mathcal{S}'$ is also a model of $\mathcal{S}$ if restricted to the class names in $\mathcal{C}$ and the attribute names in $\mathcal{A}$. ∎

Lemma 2.3.7 *Let $\mathcal{S} = (\mathcal{C}, \mathcal{A}, \mathcal{T})$ and $\mathcal{S}' = (\mathcal{C}', \mathcal{A}', \mathcal{T}')$, where $\mathcal{C} \subseteq \mathcal{C}'$ and $\mathcal{A} \subseteq \mathcal{A}'$, be equivalent schemata, and $C_1, C_2 \in \mathcal{C}$. Then the following holds:*

1. $(\mathcal{S} \not\models_u) \iff (\mathcal{S}' \not\models_u)$.
2. $(\mathcal{S} \not\models_u C_1 \equiv \bot) \iff (\mathcal{S}' \not\models_u C_1 \equiv \bot)$.
3. $(\mathcal{S} \models_u C_1 \preceq C_2) \iff (\mathcal{S}' \models_u C_1 \preceq C_2)$.
4. $(\mathcal{S} \models_u C_1 \equiv C_2) \iff (\mathcal{S}' \models_u C_1 \equiv C_2)$.

Proof. Straightforward. □

2.4 Finite versus Unrestricted Models

When defining the semantics of $\mathcal{L}$-languages and schemata we did not make any assumption on the domain of the interpretation, except that it be nonempty. Also, the reasoning services were defined in terms of arbitrary interpretations, possibly with an infinite domain.

The question of finiteness of the interpretation domain has very rarely been a matter of debate among researchers working on formalisms for structured knowledge representation. This is mainly for two reasons: First, since Description Logics are just particular kinds of logic (in many cases subsets of First Order Logic) infinite models and classes with an infinite extension are perfectly admissible. Second, and probably more relevant, the expressivity of most of the formalisms studied in the literature is such that the assumption of finiteness of the interpretation domain does not influence reasoning at all. This can be restated by saying that these formalisms have the *finite model property*, which means that if a schema admits a model at all, then it also admits one in which the interpretation domain is finite.

In Databases, on the other hand, the usual assumption is that the interpretation domain, which corresponds to a database state, is finite. The problem of distinguishing between reasoning in finite and unrestricted domains has for example been considered in the context of dependency theory, where the interaction of functional and inclusion dependencies may cause the implication of some dependencies to hold only if the database is assumed to be finite [50].

Since we are dealing with formalisms for the representation of knowledge, it is often necessary to make the assumption that the underlying interpretation is finite. Such an assumption may have important consequences on inference and completely change the set of facts that can be deduced from a given schema. Indeed, a whole branch of logic, namely *finite model theory*, is concerned with the study and characterization of properties of logics that hold only in finite structures, and is therefore of particular relevance to Computer Science [86]. Recent work has shown that there is a tight connection between the expressivity of logics when interpreted over finite structures, and complexity classes studied in computational complexity theory. Fagin gave the first characterization of a complexity class in terms of logic (i.e. independent of any notion of computation of a machine and of time), by charactering **NP** in terms of existential second order logic (see [75] for a recent presentation of these results). Successively, a great variety of similar characterizations has been discovered, and *Descriptive Complexity Theory*, as the area is called now, is a very active field of research. These issues are also of fundamental importance for the study and development of query languages in databases, where researchers are concerned with the tradeoff between the expressivity of a database query languages and the complexity of actually computing a query expressed in that language [44, 160, 99, 100, 4].

In this thesis we are also concerned with the study of properties of logics when interpreted over finite structures. We follow, however, a different line of research than the one just mentioned, and have concentrated on analyzing decidability and complexity of inference at the intensional level, with respect to both finite and unrestricted models. The following definition captures this distinction between reasoning in the two contexts.

Definition 2.4.1 Given an $\mathcal{L}$-language $\mathcal{L}$, an $\mathcal{L}$-schema $\mathcal{S}$ and two class expressions E, E' of $\mathcal{L}$, we call:

- *Finite schema consistency* the problem of establishing whether $\mathcal{S}$ is *finitely consistent* (written as $\mathcal{S} \not\models_f$), i.e. whether it admits a finite model.
- *Finite class consistency* the problem of establishing whether E is *finitely consistent in* $\mathcal{S}$ (written as $\mathcal{S} \not\models_f E \equiv \bot$), i.e. whether there is a finite model $\mathcal{I}$ of $\mathcal{S}$ such that $E^{\mathcal{I}} \neq \emptyset$.
- *Finite class subsumption* the problem of establishing whether E is *finitely subsumed by* E' *in* $\mathcal{S}$ (written as $\mathcal{S} \models_f E \preceq E'$), i.e. whether $E^{\mathcal{I}} \subseteq E'^{\mathcal{I}}$ for every finite model $\mathcal{I}$ of $\mathcal{S}$.
- *Finite class equivalence* the problem of establishing whether E is *finitely equivalent to* E' *in* $\mathcal{S}$ (written as $\mathcal{S} \models_f E \equiv E'$), i.e. whether $E^{\mathcal{I}} = E'^{\mathcal{I}}$ for every finite model $\mathcal{I}$ of $\mathcal{S}$. ∎

The distinction between finite and unrestricted reasoning is indeed necessary, since the expressivity of some $\mathcal{L}$-languages causes the finite model property to fail to hold. More precisely, the interaction of functional restrictions, inverse attributes and cycles that occurs already in primitive $\mathcal{LFI}$-schemata is sufficient to specify a class in a schema which can be populated only in an infinite model. Similarly, we can construct a primitive $\mathcal{LFI}$-schema in which one class is subsumed by another one only if we restrict the attention to finite models. This is illustrated in the following two examples.

Example 2.4.2 Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be the $\mathcal{LFI}$-schema where

$\mathcal{C} := \{\texttt{FirstGuard}, \texttt{Guard}\}$,
$\mathcal{A} := \{\texttt{shields}\}$,

and the set $\mathcal{T}$ of assertions consists of:

$$\begin{aligned} \texttt{FirstGuard} &\doteq \texttt{Guard} \sqcap \forall \texttt{shields}^{-}.\bot \\ \texttt{Guard} &\doteq \exists \texttt{shields} \sqcap \forall \texttt{shields.Guard} \sqcap \exists^{\leq 1} \texttt{shields}^{-}. \end{aligned}$$

The schema $\mathcal{S}$ states that a first guard is a guard whom nobody shields. A guard is somone who shields something and all it shields are guards. Moreover, a guard is shielded by at most one individual. It is easy to see, that the existence of a first guard implies the existence of an infinite sequence of guards, each one shielding the following one[4]. More formally, if $\mathcal{I}$ is a model of $\mathcal{S}$ and there is an instance $o_0 \in \Delta^{\mathcal{I}}$ of `FirstGuard`, then there must be an infinite sequence $o_1, o_2, \ldots \in \Delta^{\mathcal{I}}$ of distinct instances of `Guard`, such that $(o_i, o_{i+1}) \in \texttt{shields}^{\mathcal{I}}$, for $i \geq 0$. In fact, if we try to populate the class `Guard` with the least number of instances, starting with o_0, for each instance o_i introduced in $\texttt{Guard}^{\mathcal{I}}$, we are forced to introduce a new instance o_{i+1}. This is because any o_j with $0 < j \leq i$ appears already as second component in the instance (o_{j-1}, o_j) of `shields`, and therefore cannot be reused as a `shields`-successor for o_i, and because also o_0 cannot appear as second component since it is an instance of `FirstGuard`. Summing up, we can say that $\mathcal{S} \not\models_u \texttt{FirstGuard} \equiv \bot$, while $\mathcal{S} \models_f \texttt{FirstGuard} \equiv \bot$. ∎

Example 2.4.3 Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be the $\mathcal{LFI}$-schema where

$\mathcal{C} := \{\texttt{Number}, \texttt{Even}\}$,
$\mathcal{A} := \{\texttt{doubles}\}$,

and the set $\mathcal{T}$ of assertions consists of:

$$\begin{aligned} \texttt{Number} &\doteq \exists \texttt{doubles}^{-} \sqcap \forall \texttt{doubles}^{-}.\texttt{Even} \\ \texttt{Even} &\doteq \texttt{Number} \sqcap \exists^{\leq 1} \texttt{doubles} \sqcap \forall \texttt{doubles.Number}. \end{aligned}$$

Intuitively, the schema $\mathcal{S}$ states that for each number there is an even number which doubles it, and that all numbers which double it are even. Each even number is a number, doubles at most one number, and doubles only numbers. Observe that for any model $\mathcal{I}$ of $\mathcal{S}$ the universal quantifications together with the functionality of `doubles` in the assertions imply that $\sharp\texttt{Even}^{\mathcal{I}} \geq \sharp\texttt{Number}^{\mathcal{I}}$, while the direct inclusion assertion between `Even` and `Number` implies that $\sharp\texttt{Even}^{\mathcal{I}} \leq \sharp\texttt{Number}^{\mathcal{I}}$. Therefore, the two classes have the same cardinality, and since one is a subclass of the other, if the domain is finite, the two classes coincide. This does not necessarily hold for infinite domains. In fact, the names chosen for the classes and the attribute suggest already an infinite model of the schema in which `Number` and `Even` are interpreted differently. The model is obtained by taking the natural numbers as domain, and interpreting `Number` as the whole domain, `Even` as the even numbers, and `doubles` as the set $\{(2n, n) \mid n \geq 0\}$. Summarizing, we can say that $\mathcal{S} \models_f \texttt{Number} \equiv \texttt{Even}$ while $\mathcal{S} \not\models_u \texttt{Number} \equiv \texttt{Even}$. ∎

Notice that Propositions 2.3.2, 2.3.3, 2.3.4, and 2.3.5 and Lemma 2.3.7 extend easily to the case where reasoning is performed with respect to finite models only.

[4]This situation resembles the one described in Franz Kafka's novel "The Trial".

Chapter 3

Modeling Known Formalisms

In this chapter we show that $\mathcal{L}$-languages (in particular $\mathcal{LUNI}$ used with primitive schemata) and the associated reasoning capabilities represent the essential core of the class-based representation formalisms. This is done by providing translations from frames, semantic data models, and object oriented data models to $\mathcal{L}$-schemata. The expressivity of $\mathcal{L}$-languages allows us to devise these translation in such a way that the typical problems concerning reasoning in all these formalisms have a direct correspondence in reasoning on $\mathcal{L}$-schemata. The translations also show that the combination of constructs necessary to capture all structural aspects in the different contexts is exactly that of primitive $\mathcal{LUNI}$-schemata. Therefore, in order to perform intensional deductions on frames, semantic data models, and object-oriented data models we can resort to the methods developed in Chapters 5 and 6 for reasoning on primitive $\mathcal{LUNI}$-schemata.

In this sense, $\mathcal{L}$-schemata not only provide a common powerful representation tool, but may also contribute to significant developments for the languages belonging to all the three families

The chapter is organized as follows. In Section 3.1 we compare $\mathcal{L}$-languages with Description Logics, pointing out some of the aspects that are currently *not* dealt with in $\mathcal{L}$-languages. In Section 3.2 we show how to reduce reasoning in frame based systems to reasoning on $\mathcal{L}$-schemata. In Sections 3.3 and 3.4 we provide similar transformations for semantic and object-oriented models, thus showing also that reasoning in these formalisms is decidable.

3.1 Comparison with Description Logics

In the last decade, the research on structured Knowledge Representation concentrated on the definition of Description Logics (also called Concept Languages), which are subsets of first-order logics, introduced for the formalization of the KL-ONE system and its successors (see [168]). Most of the constructors of the family of $\mathcal{L}$-languages correspond in fact (both in syntax and semantics) to well known constructors considered and studied in Description Logics [120, 65, 64, 11]. A notable exception is the repeat constructor, which has not been considered before in knowledge representation formalisms as a means to define well founded structures. We should mention, that the same effect can be achieved by including fixpoint definitions in class expressions [141, 57]. In this case, however, decidability remains open if we add number restrictions and inverse roles, while in Chapter 5 we show decidability of reasoning in unrestricted models for the combination of these constructors with repeat.

Some combinations of constructors present in $\mathcal{L}$-languages have not been considered before (in particular, the combination of inverse roles and number restrictions in cyclic assertions, and repeat) and in fact result in a family of formalism that subsumes most known Description Logics. Therefore, instead of showing a correspondence we mention briefly those aspects that have been considered in Description Logics and that have no correspondence in $\mathcal{L}$-languages. The most important restriction stems from the fact that we deal only with reasoning at the intensional level, while the development of reasoning procedures that can deal with the integration of intensional and extensional reasoning constitutes an important aspect of research in Knowledge Representation [93, 14, 39, 137, 69, 139, 138, 37]. Other aspects and constructors that are not present in $\mathcal{L}$-languages and that have been subject of study include the following:

- Constructors for specifying classes by an enumeration of single objects [138, 139].

- The use of epistemic operators in class and link expressions, which allow for the formalization of certain procedural and non-monotonic mechanisms [66, 67].
- Constructors for expressing cardinality constraints on the number of instances of a certain class (as opposed to the number of instances connected through a link) [15].
- The integration of concrete domains (such as numbers) and of specialized algorithms for reasoning on these domains [13].
- Nonmonotonic extensions of Description Logics by means of default rules [17, 16].

An integration of some (or all) of these aspects with the combination of constructors that is peculiar of $\mathcal{L}$-languages, and the development of adequate procedures for reasoning on both intensional and extensional knowledge represents an interesting challenge.

3.2 Modeling Frame Based Systems

Frame languages are based on the idea of expressing knowledge by means of *frames*, which are structures representing classes of objects in terms of the properties that their instances must satisfy. Such properties are defined by the frame *slots*, that constitute the items of a frame definition. Since the 70s a large number of frame systems have been developed, with different goals and different features. Description Logics and consequently $\mathcal{L}$-languages bear a close relationship with the KL-ONE family of frame systems (see [168]). However, here we would like to consider frame systems from a more general perspective, as discussed for example in [102, 126], and establish the correspondence with $\mathcal{L}$-languages in this context.

Note that we are restricting our attention to those aspects that are related to the taxonomic structure. For this reason we do not consider those features that cannot be captured in a first-order framework, such as default values in the slots, attached procedures, and the specification of overriding inheritance policies. Some of the issues concerning the modeling of these aspects in Description Logics are addressed in [68, 71], within a modal nonmonotonic extension of Description Logics.

3.2.1 Syntax of Frame Based Systems

To make the correspondence precise, we need to fix syntax and semantics for the frame systems we consider. Since there is no accepted standard, we have chosen to use the notation adopted in [78], which is used also in the KEE[1] system.

Definition 3.2.1 A *frame knowledge base*, denoted by $\mathcal{K}$, is formed by a set of *frame names* and *slot names*, and is constituted by a set of *frame definitions* of the following form:

$$\textbf{Frame}: \ H \ \textbf{in KB} \ \mathcal{K} \quad F,$$

where F is a *frame expression*, i.e. an expression formed according to the following syntax:

$$\begin{array}{rcl} F & \longrightarrow & \textbf{SuperClasses}: \ H_1, \ldots, H_h \\ & & \textbf{MemberSlot}: \ S_1 \\ & & \quad \textbf{ValueClass}: \ E_1 \\ & & \quad \textbf{Cardinality.Min}: \ m_1 \\ & & \quad \textbf{Cardinality.Max}: \ n_1 \\ & & \ldots \\ & & \textbf{MemberSlot}: \ S_k \\ & & \quad \textbf{ValueClass}: \ E_k \\ & & \quad \textbf{Cardinality.Min}: \ m_k \\ & & \quad \textbf{Cardinality.Max}: \ n_k \end{array}$$

[1] KEE is a trademark of Intellicorp. Note that a KEE user does not directly specify her knowledge base in this notation, but is allowed to define frames interactively via the system interface.

Frame: Course **in KB** University
  MemberSlot: enrolls
    ValueClass: Student
    Cardinality.Min: 2
    Cardinality.Max: 30
  MemberSlot: taughtby
    ValueClass: (UNION GradStud Professor)
    Cardinality.Min: 1
    Cardinality.Max: 1

Frame: AdvCourse **in KB** University
  SuperClasses: Course
  MemberSlot: enrolls
    ValueClass: (INTERSECTION GradStud (NOT Undergrad))
    Cardinality.Max: 20

Frame: BasCourse **in KB** University
  SuperClasses: Course
  MemberSlot: taughtby
    ValueClass: (INTERSECTION Professor (NOT GradStud))

Frame: Professor **in KB** University

Frame: Student **in KB** University

Frame: GradStud **in KB** University
  SuperClasses: Student
  MemberSlot: degree
    ValueClass: String
    Cardinality.Min: 1
    Cardinality.Max: 1

Frame: Undergrad **in KB** University
  SuperClasses: Student

Figure 3.1: A KEE knowledge base

H and S denote frame and slot names, respectively, m and n denote positive integers, and E denotes *slot constraint*, which are specified according to:

$$\begin{aligned} E \longrightarrow\ & H \mid \\ & (\text{INTERSECTION } E_1\ E_2) \mid \\ & (\text{UNION } E_1\ E_2) \mid \\ & (\text{NOT } E) \end{aligned}$$

■

Note that we omit the specification of the sub-classes for a frame present in KEE, since it can be directly derived from the specification of the super-classes.

Example 3.2.2 Figure 3.1 shows a simple example of knowledge base expressed in the frame language we have presented. ■

3.2.2 Semantics of Frame Based Systems

To give semantics to a set of frame definitions we resort to their interpretation in terms of first-order predicate calculus (see [91]). According to such interpretation, frame names are treated as unary predicates, while slots are considered binary predicates.

A frame expression F is interpreted as a predicate logic formula $F(x)$, which has one free variable, and is constituted by the conjunction of sentences, obtained from the super-class specification and from each slot specification. In particular, for the super-classes $H_1, \ldots, H_h$ we have:

$$H_1(x) \wedge \ldots \wedge H_h(x)$$

and for a slot specification

$$\begin{array}{l}\textbf{MemberSlot}: S\\ \quad \textbf{ValueClass}: E\\ \quad \textbf{Cardinality.Min}: m\\ \quad \textbf{Cardinality.Max}: n\end{array}$$

we have

$$\begin{array}{l}\forall y.\,(S(x,y) \rightarrow E(y)) \wedge\\ \exists y_1,\ldots,y_m.\,((\bigwedge_{i\neq j} y_i \neq y_j) \wedge S(x,y_1) \wedge \cdots \wedge S(x,y_m)) \wedge\\ \forall y_1,\ldots,y_{n+1}.\,((S(x,y_1) \wedge \cdots \wedge S(x,y_{n+1})) \rightarrow \bigvee_{i\neq j} y_i = y_j),\end{array}$$

under the assumption that within one frame definition the occurrences of x refer to the same free variable. Finally the constraints on the slots are interpreted as conjunction, disjunction and negation, respectively, i.e.:

(INTERSECTION E_1 E_2)	is interpreted as	$E_1(x) \wedge E_2(x)$
(UNION E_1 E_2)	is interpreted as	$E_1(x) \vee E_2(x)$
(NOT E)	is interpreted as	$\neg E(x)$

A frame definition

$$\textbf{Frame}: H \textbf{ in KB } \mathcal{K} \quad F$$

is then considered as the universally quantified sentence of the form

$$\forall x.(H(x) \rightarrow F(x)).$$

The whole frame knowledge base $\mathcal{K}$ is considered as the conjunction of all first-order sentences corresponding to the frame definitions in $\mathcal{K}$.

Here we regard frame definitions as necessary conditions, which is commonplace in the frame systems known as *assertional* frame systems, as opposed to *definitional* systems, such as those of the KL-ONE family, where frame definitions are interpreted as necessary and sufficient conditions.

In order to enable the comparison with our formalisms for representing structured knowledge we restrict our attention to the reasoning tasks that involve the frame knowledge base, independently on the assertional knowledge, i.e. the frames instances. In [78], several reasoning services associated with frames are mentioned, such as:

- *Consistency checking*, which amounts to verifying whether a frame H is satisfiable in a knowledge base. In particular, this involves both reasoning on cardinalities and checking whether the filler of a given slot belongs to a certain frame.
- *Inheritance*, which, in our case, amounts to the ability of identifying which of the frames are more general than a given frame, sometimes called *all-super-of* (see [126]). All the properties of the more general frames are then inherited by the more specific one. Such a reasoning is therefore based on the more general ability to check the mutual relationships between frame descriptions in the knowledge base.

These reasoning services are formalized in the first-order semantics as follows.

Definition 3.2.3 Let $\mathcal{K}$ be an frame knowledge base and H a frame in $\mathcal{K}$. We say that H is *consistent in* $\mathcal{K}$ if the first-order sentence $\mathcal{K} \wedge \exists x.H(x)$ is satisfiable. Moreover, we say that a frame description F is *more general* than H in $\mathcal{K}$ if $\mathcal{K} \models \forall x.(H(x) \rightarrow F(x))$. ∎

3.2.3 Relationship between Frame Based Systems and $\mathcal{L}$-Schemata

The first-order semantics given above allows us to establish a straightforward relationship between frame languages and $\mathcal{L}$-languages, and we can define a translation from frame knowledge bases to primitive $\mathcal{LUN}$-schemata.

We first define the function θ that maps each frame expression into an $\mathcal{LUN}$-class expression as follows:

- Every frame name H is mapped into a class name $\theta(H)$.
- Every slot name S is mapped into an attribute name $\theta(S)$.

- Every slot constraint is mapped as follows

(UNION E_1 E_2)	is mapped into	$\theta(E_1) \sqcup \theta(E_2)$.
(INTERSECTION E_1 E_2)	is mapped into	$\theta(E_1) \sqcap \theta(E_2)$.
(NOT E)	is mapped into	$\neg\theta(E)$.

- Every frame expression of the form

SuperClasses : $H_1, \ldots, H_h$
MemberSlot : S_1
  ValueClass : E_1
  Cardinality.Min : m_1
  Cardinality.Max : n_1
...
MemberSlot : S_k
  ValueClass : E_k
  Cardinality.Min : m_k
  Cardinality.Max : n_k

  is mapped into the class expression

$$\theta(H_1) \sqcap \cdots \sqcap \theta(H_h) \sqcap$$
$$\forall\theta(S_1).\theta(E_1) \sqcap \exists^{\geq m_1}\theta(S_1) \sqcap \exists^{\leq n_1}\theta(S_1) \sqcap$$
$$\cdots$$
$$\forall\theta(S_k).\theta(E_k) \sqcap \exists^{\geq m_k}\theta(S_k) \sqcap \exists^{\leq n_k}\theta(S_k).$$

This mapping allows us to translate a frame knowledge base into an $\mathcal{LUN}$-schema.

Definition 3.2.4 The $\mathcal{LUN}$-schema $\theta(\mathcal{K}) := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ corresponding to the frame knowledge base $\mathcal{K}$ is obtained as follows:

- $\mathcal{C}$ consists of one class name $\theta(H)$ for each frame name H in $\mathcal{K}$.
- $\mathcal{A}$ consists of one attribute name $\theta(S)$ for each slot name S in $\mathcal{K}$.
- $\mathcal{T}$ consists of a primitive class specification

$$\theta(H) \;\dot{\preceq}\; \theta(F)$$

  for each frame definition

Frame : H **in KB** $\mathcal{K}$ F

  in $\mathcal{K}$. ∎

We illustrate the translation on a simple example.

Example 3.2.2 (cont.) The primitive $\mathcal{LUN}$-schema corresponding to the knowledge base of Figure 3.1 is $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$, where

$\mathcal{C}$:= {Course, AdvCourse, BasCourse, Professor, Student, GradStud, Undergrad, String},

$\mathcal{A}$:= {enrolls, taughtby, degree},

and the set $\mathcal{T}$ of assertions consists of:

$$\begin{aligned}
\texttt{Course} &\;\dot{\preceq}\; \forall\texttt{enrolls.Student} \sqcap \exists^{\geq 2}\texttt{enrolls} \sqcap \exists^{\leq 30}\texttt{enrolls} \sqcap \\
&\quad\; \forall\texttt{taughtby.}(\texttt{Professor} \sqcup \texttt{GradStud}) \sqcap \exists^{=1}\texttt{taughtby} \\
\texttt{AdvCourse} &\;\dot{\preceq}\; \texttt{Course} \sqcap \forall\texttt{enrolls.}(\texttt{GradStud} \sqcap \neg\texttt{Undergrad}) \sqcap \exists^{\leq 20}\texttt{enrolls} \\
\texttt{BasCourse} &\;\dot{\preceq}\; \texttt{Course} \sqcap \forall\texttt{taughtby.}(\texttt{Professor} \sqcap \neg\texttt{GradStud}) \\
\texttt{GradStud} &\;\dot{\preceq}\; \texttt{Student} \sqcap \forall\texttt{degree.String} \sqcap \exists^{=1}\texttt{degree} \\
\texttt{Undergrad} &\;\dot{\preceq}\; \texttt{Student}.
\end{aligned}$$

∎

The correctness of the translation follows from the correspondence between the set-theoretic semantics of $\mathcal{L}$-languages and the first-order interpretation of frames (see for example [70, 28]). We can observe that inverse attributes and other link forming constructors are in fact not necessary for the formalization of frames. Indeed, the possibility of referring to the inverse of a slot has been rarely considered in frame knowledge representation systems (Some exceptions are reported in [102]). Due to the absence of inverse roles, $\mathcal{LUN}$-schemata have the finite model property, and the distinction between reasoning in finite and unrestricted models is not necessary. Consequently, all the above mentioned forms of reasoning are captured by unrestricted class consistency and class subsumption in primitive $\mathcal{LUN}$-schemata. This is summarized in the following theorem.

Theorem 3.2.5 *Let $\mathcal{K}$ be a frame knowledge-base, H be a frame in $\mathcal{K}$, F be a frame description, and $\theta(\mathcal{K})$ be the translation of $\mathcal{K}$. Then the following holds:*

- *H is consistent in $\mathcal{K}$ if and only if $\theta(\mathcal{K}) \not\models_u \theta(H) \equiv \bot$.*
- *F is more general than H in $\mathcal{K}$ if and only if $\theta(\mathcal{K}) \models_u \theta(H) \preceq \theta(F)$.*

Proof. The claim trivially follows from the semantics of frame knowledge bases and the translation into $\mathcal{L}$-schemata that we have adopted. □

3.3 Modeling Semantic Data Models

Semantic data models were introduced primarily as formalisms for database schema design. They provide a means to model databases in a much richer way than traditional data models supported by Database Management Systems, and are becoming more and more important because they are adopted in most of the recent database design methodologies and Computer Aided Software Engineering tools.

The most widespread semantic data model is the Entity-Relationship (ER) model introduced in [45]. It has by now become a standard, extensively used in the design phase of commercial applications. In the commonly accepted ER notation, classes are called *entities* and are represented as boxes, whereas relationships between entities are represented as diamonds. Arrows between entities, called *ISA* relationships, represent inclusion assertions. The links between entities and relationships represent the ER-*roles*, to which number restrictions are associated. Dashed links are used whenever such restrictions are refined for more specific entities. Finally, elementary properties of entities are modeled by *attributes*, whose values belong to one of several predefined domains, such as `Integer`, `String`, or `Boolean`.

As already mentioned, the ER model does not provide constructs for expressing explicit disjointness or disjunction, but extensions of the model allow for the use of generalization hierarchies which represent a combination of these two constructs. In order to keep the presentation simple, we do not consider generalization hierarchies in the formalization we provide, although their addition would be straightforward. Similarly, we omit attributes of relations.

We show now that ER-schemata can be modeled by $\mathcal{L}$-schemata, and thus that reasoning on an ER-schema can be reduced to reasoning on the corresponding $\mathcal{L}$-schema. More precisely, we show that the language $\mathcal{LNI}$ and just primitive schemata are already sufficient to capture all relevant aspects of the ER-model. This is particularly significant, since in Chapters 5 and 6 we will provide procedures for reasoning on these types of schemata both with respect to finite and unrestricted models, thus showing that reasoning on the ER-model, and more generally on semantic data models, is decidable. We remark that if we also take generalization hierarchies into account, an ER-schema can be modeled by a primitive $\mathcal{LUNI}$-schema, for which the reasoning procedures developed in this thesis can still be applied.

In order to establish the correspondence between the two formalisms, we first need formal definitions of syntax and semantics of ER-schemata.

3.3.1 Syntax of the Entity-Relationship Model

Although the ER-model has by now become an industrial standard, several variants and extensions have been introduced, which differ in minor aspects in expressivity and in notation [45, 153, 20, 155, 156]. Also, ER-schemata are usually defined using a graphical notation which is particularly useful for an easy visualization of

the data dependencies, but which is not well suited for our purposes. Therefore we have chosen a formalization of the ER-model which abstracts with respect to the most important characteristics and allows us to show easily the correspondence to $\mathcal{L}$-schemata.

In the following, for two finite sets X and Y we call a function from a subset of X to Y an *X-labeled tuple over Y*. The labeled tuple T that maps $x_i \in X$ to $y_i \in Y$, for $i \in \{1, \ldots, k\}$, is denoted $[x_1\colon y_1, \ldots, x_k\colon y_k]$. We also write $T[x_i]$ to denote y_i.

Definition 3.3.1 An *ER-schema* is a tuple $\mathcal{S} := (\mathcal{L}_\mathcal{S}, \preceq_\mathcal{S}, att_\mathcal{S}, rel_\mathcal{S}, card_\mathcal{S})$, where

- $\mathcal{L}_\mathcal{S}$ is a finite alphabet partitioned into a set $\mathcal{E}_\mathcal{S}$ of *entity* symbols, a set $\mathcal{A}_\mathcal{S}$ of *attribute* symbols, a set $\mathcal{U}_\mathcal{S}$ of *role* symbols, a set $\mathcal{R}_\mathcal{S}$ of *relationship* symbols, and a set $\mathcal{D}_\mathcal{S}$ of *domain* symbols; each domain symbol D has an associated predefined basic domain $D^{\mathcal{B}_\mathcal{D}}$, and we assume the various basic domains to be pairwise disjoint.
- $\preceq_\mathcal{S} \subseteq \mathcal{E}_\mathcal{S} \times \mathcal{E}_\mathcal{S}$ is a binary relation over $\mathcal{E}_\mathcal{S}$.
- $att_\mathcal{S}$ is a function that maps each entity symbol in $\mathcal{A}_\mathcal{S}$ to an $\mathcal{A}_\mathcal{S}$-labeled tuple over $\mathcal{D}_\mathcal{S}$.
- $rel_\mathcal{S}$ is a function that maps each relationship symbol in $\mathcal{R}_\mathcal{S}$ to an $\mathcal{U}_\mathcal{S}$-labeled tuple over $\mathcal{E}_S$. We assume without loss of generality that:
  - Each role is specific to exactly one relationship, i.e. for two relationships $R, R' \in \mathcal{R}_\mathcal{S}$ with $R \neq R'$, if $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$ and $rel_\mathcal{S}(R') := [U'_1\colon E'_1, \ldots, U'_{k'}\colon E'_{k'}]$, then $\{U_1, \ldots, U_k\}$ and $\{U'_1, \ldots, U'_{k'}\}$ are disjoint.
  - For each role $U \in \mathcal{U}_\mathcal{S}$ there is a relationship R and an entity E such that $rel_\mathcal{S}(R) := \langle \ldots, U\colon E, \ldots \rangle$.
- $card_\mathcal{S}$ is a function from $\mathcal{E}_\mathcal{S} \times \mathcal{R}_\mathcal{S} \times \mathcal{U}_\mathcal{S}$ to $\mathbb{N}_0 \times (\mathbb{N}_0 \cup \{\infty\})$ that satisfies the following condition: for a relationship $R \in \mathcal{R}_\mathcal{S}$ such that $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$, $card_\mathcal{S}(E, R, U)$ is defined only if $U = U_i$ for some $i \in \{1, \ldots, k\}$, and if $E \preceq^*_\mathcal{S} E_i$[2]. The first component of $card_\mathcal{S}(E, R, U)$ is denoted with $cmin_\mathcal{S}(E, R, U)$ and the second component with $cmax_\mathcal{S}(E, R, U)$. If not stated otherwise, $cmin_\mathcal{S}(E, R, U)$ is assumed to be 0 and $cmax_\mathcal{S}(E, R, U)$ is assumed to be ∞. ∎

Before specifying the formal semantics of ER-schemata we give an intuitive description of the components of a schema. The relation $\preceq_\mathcal{S}$ models the ISA-relationship between entities. We do not need to make any special assumption on the form of $\preceq_\mathcal{S}$ such as acyclicity or injectivity. The function $att_\mathcal{S}$ is used to model attributes of entities. If for example $att_\mathcal{S}$ associates the $\mathcal{A}_\mathcal{S}$-labeled tuple $[A_1\colon \texttt{Integer}, A_2\colon \texttt{String}]$ to an entity E, then E has two attributes A_1, A_2 whose values are integers and strings respectively. For simplicity we assume attributes to be single-valued and mandatory, but we could easily handle also multi-valued attributes with associated cardinalities. The function $rel_\mathcal{S}$ associates to each relationship symbol R a set of roles, determining implicitly also the arity of the relationship, and for each role U in such set a distinguished entity, called the *primary entity for U in R*. In a database satisfying the schema only instances of the primary entity are allowed to participate in the relationship via the role U. The function $card_\mathcal{S}$ specifies *cardinality constraints*, i.e. constraints on the minimum and maximum number of times an instance of an entity may participate in a relationship via some role. Since such constraints are meaningful only if the entity can effectively participate in the relationship, the function is defined only for the sub-classes of the primary class. The special value ∞ is used when no restriction is posed on the maximum cardinality. Such constraints can be used to specify both existence dependencies and functionality of relations [49]. They are often used only in a restricted form, where the minimum cardinality is either 0 or 1 and the maximum cardinality is either 1 or ∞. Cardinality constraints in the form considered here have been introduced already in [5] and successively studied in [83, 111, 77, 169]. In [155] a very general form of cardinality constraints for the ER-model is formalized and deduction is studied for restricted cases.

Example 3.3.2 Figure 3.2 shows a simple ER-schema modeling a similar state of affairs as the KEE knowledge base in Figure 3.1. We have used the standard graphic notation for ER-schemata, except for the dashed link, which represents the refinement of a cardinality constraint for the participation of a sub-class (in our case `AdvCourse`) in a relationship (`ENROLLING`). ∎

[2] $\preceq^*_\mathcal{S}$ denotes the reflexive transitive closure of $\preceq_\mathcal{S}$.

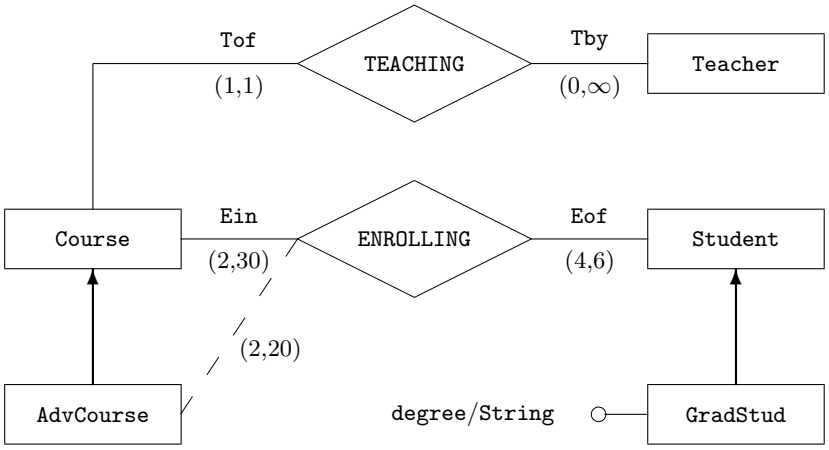

Figure 3.2: An ER-schema

3.3.2 Semantics of the Entity-Relationship Model

The semantics of an ER-schema can be given by specifying which database states conform to the information structure represented by the schema. Formally, a database state $\mathcal{B}$ corresponding to an ER-schema $\mathcal{S} := (\mathcal{L}_\mathcal{S}, \preceq_\mathcal{S}, att_\mathcal{S}, rel_\mathcal{S}, card_\mathcal{S})$ is constituted by a nonempty *finite* set $\Delta^\mathcal{B}$, assumed to be disjoint from all basic domains, and a function $\cdot^\mathcal{B}$ that maps

- every domain symbol $D \in \mathcal{D}_\mathcal{S}$ to the corresponding basic domain $D^{\mathcal{B}_\mathcal{D}}$,
- every entity $E \in \mathcal{E}_\mathcal{S}$ to a subset $E^\mathcal{B}$ of $\Delta^\mathcal{B}$,
- every attribute $A \in \mathcal{A}_\mathcal{S}$ to a set $A^\mathcal{B} \subseteq \Delta^\mathcal{B} \times \bigcup_{D \in \mathcal{D}_\mathcal{S}} D^{\mathcal{B}_\mathcal{D}}$, and
- every relationship $R \in \mathcal{R}_\mathcal{S}$ to a set $R^\mathcal{B}$ of $\mathcal{U}_\mathcal{S}$-labeled tuples over $\Delta^\mathcal{B}$.

The elements of $E^\mathcal{B}$, $A^\mathcal{B}$, and $R^\mathcal{B}$ are called *instances* of E, A, and R respectively.

A database state is considered acceptable if it satisfies all integrity constraints that are part of the schema. This is captured by the definition of legal database state.

Definition 3.3.3 A database state $\mathcal{B}$ is said to be *legal for* an ER-schema $\mathcal{S} := (\mathcal{L}_\mathcal{S}, \preceq_\mathcal{S}, att_\mathcal{S}, rel_\mathcal{S}, card_\mathcal{S})$, if it satisfies the following conditions:

- For each pair of entities $E_1, E_2 \in \mathcal{E}_\mathcal{S}$ such that $E_1 \preceq_\mathcal{S} E_2$, it holds that $E_1^\mathcal{B} \subseteq E_2^\mathcal{B}$.
- For each entity $E \in \mathcal{E}_\mathcal{S}$, if $att_\mathcal{S}(E) := [A_1\colon D_1, \ldots, A_h\colon D_h]$, then for each instance $e \in E^\mathcal{B}$ and for each $i \in \{1, \ldots, h\}$ the following holds:
  - there is exactly one element $a_i \in A_i^\mathcal{B}$ whose first component is e, and
  - the second component of a_i is an element of $D^{\mathcal{B}_\mathcal{D}}$.
- For each relationship $R \in \mathcal{R}_\mathcal{S}$ such that $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$, all instances of R are of the form $[U_1\colon e_1, \ldots, U_k\colon e_k]$, where $e_i \in E_i^\mathcal{B}$, $i \in \{1, \ldots, k\}$.
- For each relationship $R \in \mathcal{R}_\mathcal{S}$ such that $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$, for each $i \in \{1, \ldots, k\}$, for each entity $E \in \mathcal{E}_\mathcal{S}$ such that $E \preceq_\mathcal{S}^* E_i$ and for each instance e of E in $\mathcal{I}$, it holds that

$$cmin_\mathcal{S}(E, R, U_i) \leq \sharp\{r \in R^\mathcal{B} \mid r[U_i] = e\} \leq cmax_\mathcal{S}(E, R, U_i).$$

■

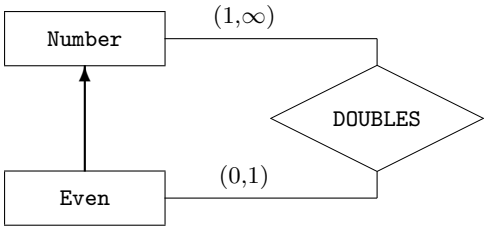

Figure 3.3: The ER-schema corresponding to Example 2.4.3

Notice that the definition of database state reflects the usual assumption in the whole database area that database states are finite structures (see also [50]). In fact, the basic domains are not required to be finite, but for each legal database state for a schema, only a finite set of values from the basic domains are actually of interest. We define the *active domain* $\Delta^{\mathcal{B}}_{act}$ of a database state $\mathcal{B}$ as the set of all elements of the basic domains $D^{\mathcal{B}_{\mathcal{D}}}$, $D \in \mathcal{D}_{\mathcal{S}}$, that effectively appear as values of attributes in $\mathcal{B}$. More formally:

$$\Delta^{\mathcal{B}}_{act} := \{d \in D^{\mathcal{B}_{\mathcal{D}}} \mid D \in \mathcal{D}_{\mathcal{S}} \wedge \exists A \in \mathcal{A}_{\mathcal{S}}, e \in \Delta^{\mathcal{B}} \colon (e,d) \in A^{\mathcal{B}}\}.$$

Since $\Delta^{\mathcal{B}}$ is finite and $\mathcal{A}_{\mathcal{S}}$ contains only a finite number of attributes which are functional by definition, also $\Delta^{\mathcal{B}}_{act}$ is finite.

Reasoning in the ER-model includes verifying entity satisfiability and deducing inheritance. *Entity satisfiability* amounts to checking whether a given entity can be populated in some legal database state (see [7, 111, 59]), and corresponds to the notion of class consistency in $\mathcal{L}$-schemata. Deducing *inheritance* amounts to verifying whether in all databases that are legal for the schema, every instance of an entity is also an instance of another entity. Such implied ISA relationships can arise for different reasons. Either trivially, from the transitive closure of the explicit ISA relationships present in the schema, or in more subtle ways, through specific patterns of cardinality restrictions along cycles in the schema and the requirement of the database state to be finite [111, 50].

Example 3.3.4 Figure 3.3 shows an ER-schema modeling the same situation as the $\mathcal{L}$-schema of Example 2.4.3. Arguing exactly as in that example we can conclude that the two entities `Number` and `Even` denote the same set of objects in every finite database legal for the schema, although the ISA relation from `Number` to `Even` is not stated explicitly. It is implied, however, due to the cycle involving the relationship and the two classes and due to the particular form of cardinality constraints. ∎

3.3.3 Relationship between ER-Schemata and $\mathcal{L}$-Schemata

We show now that the different forms of reasoning on ER-schemata are captured by finite class consistency and finite class subsumption in primitive $\mathcal{LNI}$-schemata. The correspondence between the ER-model and our class based formalism is established again by defining a translation ϕ from ER-schemata to $\mathcal{L}$-schemata, and then proving that there is a correspondence between legal database states and finite models of the derived schema.

Definition 3.3.5 Let $\mathcal{S} := (\mathcal{L}_{\mathcal{S}}, \preceq_{\mathcal{S}}, att_{\mathcal{S}}, rel_{\mathcal{S}}, card_{\mathcal{S}})$ be an ER-schema. The primitive $\mathcal{LNI}$-schema $\phi(\mathcal{S}) := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ is defined as follows:
The set $\mathcal{C}$ of classes of $\phi(\mathcal{S})$ contains the following elements:

- For each domain symbol $D \in \mathcal{D}_{\mathcal{S}}$, a class name $\phi(D)$.
- For each entity $E \in \mathcal{E}_{\mathcal{S}}$, a class name $\phi(E)$.
- For each relationship $R \in \mathcal{R}_{\mathcal{S}}$, a class name $\phi(R)$.

The set $\mathcal{A}$ of attributes of $\phi(\mathcal{S})$ contains the following elements:

- For each attribute $A \in \mathcal{A}_{\mathcal{S}}$, an attribute name $\phi(A)$.
- For each relationship $R \in \mathcal{R}_{\mathcal{S}}$ such that $rel_{\mathcal{S}}(R) := [U_1 \colon E_1, \ldots, U_k \colon E_k]$, k attribute names $\phi(U_1), \ldots, \phi(U_k)$.

The set $\mathcal{T}$ of assertions of $\phi(\mathcal{S})$ contains the following elements:

- For each pair of entities $E_1, E_2 \in \mathcal{E}_\mathcal{S}$ such that $E_1 \preceq_\mathcal{S} E_2$, the assertion
$$\phi(E_1) \;\dot{\preceq}\; \phi(E_2). \tag{3.1}$$
- For each entity $E \in \mathcal{E}_\mathcal{S}$ such that $att_\mathcal{S}(E) := [A_1\colon D_1, \ldots, A_h\colon D_h]$, the assertion
$$\phi(E) \;\dot{\preceq}\; \forall\phi(A_1).\phi(D_1) \sqcap \cdots \sqcap \forall\phi(A_h).\phi(D_h) \sqcap \exists^{=1}\phi(A_1) \sqcap \cdots \sqcap \exists^{=1}\phi(A_h). \tag{3.2}$$
- For each relationship $R \in \mathcal{R}_\mathcal{S}$ such that $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$, the assertions
$$\phi(R) \;\dot{\preceq}\; \forall\phi(U_1).\phi(E_1) \sqcap \cdots \sqcap \forall\phi(U_k).\phi(E_k) \sqcap \exists^{=1}\phi(U_1) \sqcap \cdots \sqcap \exists^{=1}\phi(U_k) \tag{3.3}$$
$$\phi(E_i) \;\dot{\preceq}\; \forall(\phi(U_i))^{-1}.\phi(R), \qquad i \in \{1, \ldots, k\}. \tag{3.4}$$
- For each relationship $R \in \mathcal{R}_\mathcal{S}$ such that $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$, for $i \in \{1, \ldots, k\}$, and for each entity $E \in \mathcal{E}_\mathcal{S}$ such that $E \preceq^*_\mathcal{S} E_i$,
  - if $m := cmin_\mathcal{S}(E, R, U_i) \neq 0$, the assertion
  $$\phi(E) \;\dot{\preceq}\; \exists^{\geq m}(\phi(U_i))^{-1}. \tag{3.5}$$
  - if $n := cmax_\mathcal{S}(E, R, U_i) \neq \infty$, the assertion
  $$\phi(E) \;\dot{\preceq}\; \exists^{\leq n}(\phi(U_i))^{-1}. \tag{3.6}$$
- For each pair of symbols $X_1, X_2 \in \mathcal{E}_\mathcal{S} \cup \mathcal{R}_\mathcal{S} \cup \mathcal{D}_\mathcal{S}$ such that $X_1 \neq X_2$ and $X_1 \in \mathcal{R}_\mathcal{S} \cup \mathcal{D}_\mathcal{S}$, the assertion
$$\phi(X_1) \;\dot{\preceq}\; \neg\phi(X_2). \tag{3.7}$$

■

We observe that, by means of the inverse constructor, a binary relationship could be treated in a simpler way by choosing a traversal direction and mapping the relationship directly to an $\mathcal{LNI}$-attribute. Also, the constructors for building tuples and tuples with keys introduced in Section 2.1.8 could have been adopted to represent arbitrary relations. These constructors, together with the constructor for sets can be used to capture also more complex semantic data models, as all those discussed in [98]. Our objective here was, however, to illustrate the minimal requirements on $\mathcal{L}$-schemata needed for establishing the correspondence, and for the ER-model primitive $\mathcal{LNI}$-schemata have already the required expressivity, as shown below.

Again, we illustrate the translation on an example.

Example 3.3.2 (cont.) The primitive $\mathcal{LNI}$-schema that captures exactly the semantics of the ER-schema of Figure 3.2 is $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$, where

$\mathcal{C}$:= {Course, AdvCourse, Teacher, Student, GradStud, TEACHING, ENROLLING, String},

$\mathcal{A}$:= {Tof, Tby, Ein, Eof, degree},

and the set $\mathcal{T}$ of assertions consists of:

$$\begin{aligned}
\texttt{TEACHING} &\;\dot{\preceq}\; \forall\texttt{Tof.Course} \sqcap \exists^{=1}\texttt{Tof} \sqcap \forall\texttt{Tby.Teacher} \sqcap \exists^{=1}\texttt{Tby}\\
\texttt{ENROLLING} &\;\dot{\preceq}\; \forall\texttt{Ein.Course} \sqcap \exists^{=1}\texttt{Ein} \sqcap \forall\texttt{Eof.Student} \sqcap \exists^{=1}\texttt{Eof}\\
\texttt{Course} &\;\dot{\preceq}\; \forall\texttt{Tof}^-\texttt{.TEACHING} \sqcap \exists^{=1}\texttt{Tof}^- \sqcap \forall\texttt{Ein}^-\texttt{.ENROLLING} \sqcap \exists^{\geq 2}\texttt{Ein}^- \sqcap \exists^{\leq 30}\texttt{Ein}^-\\
\texttt{AdvCourse} &\;\dot{\preceq}\; \texttt{Course} \sqcap \exists^{\leq 20}\texttt{Ein}^-\\
\texttt{Teacher} &\;\dot{\preceq}\; \forall\texttt{Tby}^-\texttt{.TEACHING}\\
\texttt{Student} &\;\dot{\preceq}\; \forall\texttt{Eof}^-\texttt{.ENROLLING} \sqcap \exists^{\geq 4}\texttt{Eof}^- \sqcap \exists^{\leq 6}\texttt{Eof}^-\\
\texttt{GradStud} &\;\dot{\preceq}\; \texttt{Student} \sqcap \forall\texttt{degree.String} \sqcap \exists^{=1}\texttt{degree}.
\end{aligned}$$

■

The translation demonstrates that both inverse attributes and number restrictions are necessary in order to capture the semantics of ER-schemata. Notice also that the assumption of acyclicity of the resulting $\mathcal{L}$-schema is unrealistic in this case, and in order to exploit the correspondence for reasoning in the ER-model, we need techniques that can deal with inverse attributes, number restrictions, and cyclic schemata together. As shown in Examples 2.4.2 and 2.4.3, the combination of these factors causes the finite model property to fail to hold, and we need to resort to reasoning methods for finite models.

In fact, we can reduce reasoning in the ER-model to finite class consistency and finite class subsumption in primitive $\mathcal{LNI}$-schemata. For this purpose we define a mapping between database states corresponding to an ER-schema and finite interpretations of the $\mathcal{L}$-schema derived from it. Due to the possible presence of relations with arity greater than 2, this mapping is however not one-to-one and we first need to characterize those interpretations of the $\mathcal{L}$-schema that directly correspond to database states.

Definition 3.3.6 Let $\mathcal{S} := (\mathcal{L}_\mathcal{S}, \preceq_\mathcal{S}, att_\mathcal{S}, rel_\mathcal{S}, card_\mathcal{S})$ be an ER-schema and $\phi(\mathcal{S})$ be defined as above. An interpretation $\mathcal{I}$ of $\phi(\mathcal{S})$ is *relation-descriptive*, if for every relationship $R \in \mathcal{R}_\mathcal{S}$, with $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$, for every $o, o' \in (\phi(R))^\mathcal{I}$, we have that

$$(\forall o'' \in \Delta^\mathcal{I}\colon \bigwedge_{1 \le i \le k} ((o, o'') \in (\phi(U_i))^\mathcal{I} \leftrightarrow (o', o'') \in (\phi(U_i))^\mathcal{I})) \rightarrow o = o'.$$

Intuitively, the extension of a relationship in a database state is a *set* of labeled tuples, and such set does not contain the same element twice. Therefore it is implicit in the semantics that there cannot be two labeled tuples connected through all roles of the relationship to exactly the same elements of the domain. In a model of the $\mathcal{L}$-schema corresponding to the ER-schema, on the other hand, each tuple is represented by a new object, and the above condition is not implicit anymore. It is easy to see that it also cannot be imposed by suitable assertions, except in the case when the relationship is binary. The following lemma, however, shows that we do not need such an explicit condition, when we are interested in reasoning on an $\mathcal{L}$-schema corresponding to an ER-schema. This is due to the fact that we can always restrict ourselves to considering only relation-descriptive models.

Lemma 3.3.7 *Let $\mathcal{S}$ be an ER-schema and $\phi(\mathcal{S})$ be the $\mathcal{LNI}$-schema obtained from $\mathcal{S}$ according to Definition 3.3.5. Let further E be a class expression of $\phi(\mathcal{S})$. If E is finitely consistent in $\phi(\mathcal{S})$, then there is a finite relation-descriptive model $\mathcal{I}$ of $\phi(\mathcal{S})$ such that $E^\mathcal{I} \neq \emptyset$.*

Proof. The claim follows from Lemma 5.2.4, by observing that the construction in the proof preserves finiteness of the model and class consistency. □

Using this result it is possible to establish a correspondence between database states legal for an ER-schema and relation-descriptive models of the resulting $\mathcal{LNI}$-schema.

Proposition 3.3.8 *For every ER-schema $\mathcal{S} := (\mathcal{L}_\mathcal{S}, \preceq_\mathcal{S}, att_\mathcal{S}, rel_\mathcal{S}, card_\mathcal{S})$ there exist two mappings $\alpha_\mathcal{S}$, from database states corresponding to $\mathcal{S}$ to finite interpretations of its translation $\phi(\mathcal{S})$, and $\beta_\mathcal{S}$, from finite relation-descriptive interpretations of $\phi(\mathcal{S})$ to database states corresponding to $\mathcal{S}$, such that:*

1. *For each database state $\mathcal{B}$ legal for $\mathcal{S}$, $\alpha_\mathcal{S}(\mathcal{B})$ is a finite model of $\phi(\mathcal{S})$, and for each symbol $X \in \mathcal{E}_\mathcal{S} \cup \mathcal{A}_\mathcal{S} \cup \mathcal{R}_\mathcal{S} \cup \mathcal{D}_\mathcal{S}$, $X^\mathcal{B} = (\phi(X))^{\alpha_\mathcal{S}(\mathcal{B})}$.*
2. *For each finite relation-descriptive model $\mathcal{I}$ of $\phi(\mathcal{S})$, $\beta_\mathcal{S}(\mathcal{I})$ is a database state legal for $\mathcal{S}$, for each entity $E \in \mathcal{E}_\mathcal{S}$, $(\phi(E))^\mathcal{I} = E^{\beta_\mathcal{S}(\mathcal{I})}$, and for each symbol $X \in \mathcal{A}_\mathcal{S} \cup \mathcal{R}_\mathcal{S} \cup \mathcal{D}_\mathcal{S}$, $\sharp\phi(X)^\mathcal{I} = \sharp X^{\beta_\mathcal{S}(\mathcal{I})}$, and $\beta_\mathcal{S}((\phi(X))^\mathcal{I}) = X^{\beta_\mathcal{S}(\mathcal{I})}$.*

Proof. (1) Given a database state $\mathcal{B}$ we define the interpretation $\mathcal{I} := \alpha_\mathcal{S}(\mathcal{B})$ of $\phi(\mathcal{S})$ as follows:

- $\Delta^\mathcal{I} := \Delta^\mathcal{B} \cup \Delta^\mathcal{B}_{act} \cup \bigcup_{R \in \mathcal{R}_\mathcal{S}} R^\mathcal{B}$.
- For each symbol $X \in \mathcal{E}_\mathcal{S} \cup \mathcal{A}_\mathcal{S} \cup \mathcal{R}_\mathcal{S} \cup \mathcal{D}_\mathcal{S}$,

$$(\phi(X))^\mathcal{I} := X^\mathcal{B}. \tag{3.8}$$

- For each relationship $R \in \mathcal{R}_\mathcal{S}$ such that $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$,

$$(\phi(U_i))^\mathcal{I} := \{(r, e) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \mid r \in R^\mathcal{B}, \text{ and } r[U_i] = e\}, \quad i \in \{1, \ldots, k\}. \tag{3.9}$$

Let $\mathcal{B}$ be a legal database state. To prove claim (1) it is sufficient to show that $\mathcal{I}$ satisfies every assertion in $\phi(\mathcal{S})$. Assertions 3.1 are satisfied since $\mathcal{B}$ satisfies the set inclusion between the extensions of the corresponding entities. With respect to assertions 3.2, let $E \in \mathcal{E}_\mathcal{S}$ be an entity such that $att_\mathcal{S}(E) := [A_1\colon D_1, \ldots, A_h\colon D_h]$, and consider an instance $e \in (\phi(E))^\mathcal{I}$. We have to show that for each $i \in \{1, \ldots, h\}$, there is exactly one element $e_i \in \Delta^\mathcal{I}$ such that $(e, e_i) \in (\phi(A_i))^\mathcal{I}$, and moreover that $e_i \in (\phi(D_i))^\mathcal{I}$. By 3.8, $e \in E^\mathcal{B}$, and by definition of legal database state there is exactly one element $a_i \in A_i^\mathcal{B} = (\phi(A_i))^\mathcal{I}$ whose first component is e. Moreover, the second component e_i of a_i is an element of $D_i^{\mathcal{B}_\mathcal{D}} = (\phi(D_i))^\mathcal{I}$. With respect to assertions 3.3, let $R \in \mathcal{R}_\mathcal{S}$ be a relationship such that $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$, and consider an instance $r \in (\phi(R))^\mathcal{I}$. We have to show that for each $i \in \{1, \ldots, k\}$ there is exactly one element $e_i \in \Delta^\mathcal{I}$ such that $(r, e_i) \in (\phi(U_i))^\mathcal{I}$, and that moreover $e_i \in (\phi(E_i))^\mathcal{I}$. By 3.8, $r \in R^\mathcal{B}$, and by definition of legal database state, r is a labeled tuple of the form $[U_1\colon e'_1, \ldots, U_k\colon e'_k]$, where $e'_i \in E_i^\mathcal{B}$, $i \in \{1, \ldots, k\}$. Therefore r is a function defined on $\{U_1, \ldots, U_k\}$, and by 3.9, e_i is unique and equal to e'_i. Moreover, again by 3.8, $e_i \in (\phi(E_i))^\mathcal{I} = E_i^\mathcal{B}$. Assertions 3.4 are satisfied, since by 3.9 the first component of each element of $(\phi(U_i))^\mathcal{I}$ is always an element of $R^\mathcal{B} = (\phi(R))^\mathcal{I}$. With respect to assertions 3.5, let $R \in \mathcal{R}_\mathcal{S}$ be a relationship such that $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$, let $E \in \mathcal{E}_\mathcal{S}$ be an entity such that $E \preceq_\mathcal{S} E_i$, for some $i \in \{1, \ldots, k\}$, and such that $m := cmin_\mathcal{S}(E, R, U_i) \neq 0$. Consider an instance $e \in (\phi(E))^\mathcal{I}$. We have to show that there are at least m pairs in $(\phi(U_i))^\mathcal{I}$ that have e as their second component. Since assertions 3.4 are satisfied we know that the first component of all such pairs is an instance of $\phi(R)$. By 3.8 and by definition of legal database state, there are at least m labeled tuples in $R^\mathcal{B}$ whose U_i component is equal to e. By 3.9, $(\phi(U_i))^\mathcal{I}$ contains at least m pairs whose second component is equal to e. With respect to assertions 3.6 we can proceed in a similar way. Finally, assertions 3.7 are satisfied since first, by definition the basic domains are pairwise disjoint and disjoint from $\Delta^\mathcal{B}$ and from the set of labeled tuples, second, no element of $\Delta^\mathcal{B}$ is a labeled tuple, and third, labeled tuples corresponding to different relationships cannot be equal since they are defined over different sets of roles.

(2) Let $\mathcal{I}$ be a finite relation-descriptive interpretation of $\phi(\mathcal{S})$. For each basic domain $D \in \mathcal{D}_\mathcal{S}$, let $\beta_\mathcal{S}^D$ be an injective total function from $\Delta^\mathcal{I}$ to $D^{\mathcal{B}_\mathcal{D}}$. Since $\Delta^\mathcal{I}$ is finite and each basic domain contains a countable number of elements, such a function always exists. In order to define $\beta_\mathcal{S}$ we first specify how it is applied to an element $o \in \Delta^\mathcal{I}$:

- If $o \in (\phi(E))^\mathcal{I}$ for some entity $E \in \mathcal{E}_\mathcal{S}$, then $\beta_\mathcal{S}(o) := o$.
- If $o \in (\phi(R))^\mathcal{I}$ for some relationship $R \in \mathcal{R}_\mathcal{S}$ with $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$, and there are elements $e_1, \ldots, e_k \in \Delta^\mathcal{I}$ such that $(o, e_i) \in (\phi(U_i))^\mathcal{I}$, for $i \in \{1, \ldots, k\}$, then $\beta_\mathcal{S}(o) := [U_1\colon e_1, \ldots, U_k\colon e_k]$.
- If $o \in (\phi(D))^\mathcal{I}$ for some basic domain $D \in \mathcal{D}_\mathcal{S}$, then $\beta_\mathcal{S}(o) := \beta_\mathcal{S}^D(o)$.
- Otherwise $\beta_\mathcal{S}(o) := o$.

For a pair of elements $(o_1, o_2) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I}$, $\beta_\mathcal{S}((o_1, o_2)) := (\beta_\mathcal{S}(o_1), \beta_\mathcal{S}(o_2))$, and for a set X, $\beta_\mathcal{S}(X) := \{\beta_\mathcal{S}(x) \mid x \in X\}$.

If $\mathcal{I}$ is a model of $\phi(\mathcal{S})$ the above rules define $\beta_\mathcal{S}(o)$ for every $o \in \Delta^\mathcal{I}$. Indeed, by the assertions 3.7, each $o \in \Delta^\mathcal{I}$ can be an instance of at most one class name corresponding to a relationship or basic domain, and if this is the case it is not an instance of any class name corresponding to an entity. Moreover, if $o \in (\phi(R))^\mathcal{I}$ for some relationship $R \in \mathcal{R}_\mathcal{S}$ with $rel_\mathcal{S}(R) := [U_1\colon E_1, \ldots, U_k\colon E_k]$, then by assertions 3.3, for each $i \in \{1, \ldots, k\}$ there is exactly one element $e_i \in \Delta^\mathcal{I}$ such that $(o, e_i) \in (\phi(U_i))^\mathcal{I}$. If $\mathcal{I}$ is not a model of $\phi(\mathcal{S})$ and for some $o \in \Delta^\mathcal{I}$, $\beta_\mathcal{S}(o)$ is not uniquely determined, then we choose nondeterministically one possible value.

We can now define the database state $\mathcal{B} := \beta_\mathcal{S}(\mathcal{I})$ corresponding to $\mathcal{I}$:

- $\Delta^\mathcal{B} := \Delta^\mathcal{I} \setminus \big(\bigcup_{R \in \mathcal{R}_\mathcal{S}} (\phi(R))^\mathcal{I} \cup \bigcup_{D \in \mathcal{D}_\mathcal{S}} (\phi(D))^\mathcal{I}\big)$.
- For each symbol $X \in \mathcal{E}_\mathcal{S} \cup \mathcal{A}_\mathcal{S} \cup \mathcal{R}_\mathcal{S} \cup \mathcal{D}_\mathcal{S}$, $X^\mathcal{B} := \beta_\mathcal{S}((\phi(X))^\mathcal{I})$.

It is not difficult to see, that if $\mathcal{I}$ is a model of $\phi(\mathcal{S})$, then $\mathcal{B}$ defined in such a way is a legal database state for $\mathcal{S}$ with active domain $\bigcup_{D \in \mathcal{D}_\mathcal{S}} (\phi(D))^\mathcal{I}$. □

The following theorem allows us to reduce reasoning on ER-schemata to finite model reasoning on $\mathcal{LNI}$-schemata.

Theorem 3.3.9 *Let $\mathcal{S}$ be an ER-schema, E, E' be two entities in $\mathcal{S}$, and $\phi(\mathcal{S})$ be the translation of $\mathcal{S}$. Then the following holds:*

1. *E is satisfiable in $\mathcal{S}$ if and only if $\phi(\mathcal{S}) \not\models_f \phi(E) \equiv \bot$.*
2. *E inherits from E' in $\mathcal{S}$ if and only if $\phi(\mathcal{S}) \models_f \phi(E) \preceq \phi(E')$.*

Proof. (1) "⇒" Let $\mathcal{B}$ be a legal database state with $E^{\mathcal{B}} \neq \emptyset$. By part 1 of Proposition 3.3.8, $\alpha_{\mathcal{S}}(\mathcal{B})$ is a finite model of $\phi(\mathcal{S})$ in which $(\phi(E))^{\alpha_{\mathcal{S}}(\mathcal{B})} \neq \emptyset$.

"⇐" Let $\phi(E)$ be finitely consistent in $\phi(\mathcal{S})$. By Lemma 3.3.7 there is a finite relation-descriptive model $\mathcal{I}$ of $\phi(\mathcal{S})$ with $\phi(E)^{\mathcal{I}} \neq \emptyset$. By part 2 of Proposition 3.3.8, $\beta_{\mathcal{S}}(\mathcal{I})$ is a database state legal for $\mathcal{S}$ in which $E^{\mathcal{B}} \neq \emptyset$.

(2) "⇒" Let $\phi(\mathcal{S}) \not\models_f \phi(E) \preceq \phi(E')$. By Proposition 2.3.2, $\phi(E) \sqcap \neg\phi(E')$ is finitely consistent in $\phi(\mathcal{S})$. By Lemma 3.3.7 there is a finite relation-descriptive model $\mathcal{I}$ of $\phi(\mathcal{S})$ with $o \in (\phi(E))^{\mathcal{I}}$ and $o \notin (\phi(E'))^{\mathcal{I}}$, for some $o \in \Delta^{\mathcal{I}}$. By part 2 of Proposition 3.3.8, $\beta_{\mathcal{S}}(\mathcal{I})$ is a database state legal for $\mathcal{S}$ in which $o \in E^{\mathcal{B}}$ and $o \notin E'^{\mathcal{B}}$. It follows that E does not inherit from E'.

"⇐" Assume E does not inherit from E'. Then there is a database state $\mathcal{B}$ legal for $\mathcal{S}$ where for an instance $e \in E^{\mathcal{B}}$ we have $e \notin E'^{\mathcal{B}}$. By part 1 of Proposition 3.3.8, $\alpha_{\mathcal{S}}(\mathcal{B})$ is a finite model of $\phi(\mathcal{S})$ in which $(\phi(E) \sqcap \neg\phi(E'))^{\alpha_{\mathcal{S}}(\mathcal{B})} \neq \emptyset$. By Proposition 2.3.2, $\phi(\mathcal{S}) \not\models_f \phi(E) \preceq \phi(E')$. □

3.4 Modeling Object-Oriented Data Models

Object-oriented data models have been proposed with the goal of devising database formalisms that could be integrated with object-oriented programming systems (see [104]). They are the subject of an active area of research in the Database field, and are based on the following features: (a) In contrast to traditional data models which are value-oriented, they rely on the notion of object identifiers at the extensional level, and on the notion of class at the intensional level. (b) The structure of the classes is specified by means of typing and inheritance.

3.4.1 Syntax of an Object-Oriented Model

Again, for the purpose of illustrating the correspondence with $\mathcal{L}$-schemata, we define a simple object-oriented language in the style of most popular models featuring complex objects and object identity. Although we do not refer to any specific formalism, our model is inspired by the one presented in [2]. Again, we remind that we restrict our attention to the structural component of object-oriented models and do not consider those aspects related to the definition of methods associated to the classes. Nevertheless, we argue that general techniques for schema level reasoning, as those developed in this thesis for type consistency and type inference, could be profitably exploited also for restricted forms of reasoning on methods [3].

Definition 3.4.1 An *object-oriented schema* is a tuple $\mathcal{S} := (\mathcal{C}_{\mathcal{S}}, \mathcal{A}_{\mathcal{S}}, \mathcal{D}_{\mathcal{S}})$, where:

- $\mathcal{C}_{\mathcal{S}}$ is a finite set of *class names*, denoted by the letter C.
- $\mathcal{A}_{\mathcal{S}}$ is a finite set of *attribute names*, denoted by the letter A.
- $\mathcal{D}_{\mathcal{S}}$ is a finite set of *class declarations* of the form

$$\underline{\text{Class}}\ C\ \underline{\text{is-a}}\ C_1, \ldots, C_k\ \underline{\text{type-is}}\ T,$$

in which T denotes a *type expression* built according to the following syntax:

$$\begin{aligned} T \quad \longrightarrow \quad & C \mid \\ & \underline{\text{Union}}\ T_1, \ldots, T_k\ \underline{\text{End}} \mid \\ & \underline{\text{Set-of}}\ T \mid \\ & \underline{\text{Record}}\ A_1{:}\,T_1, \ldots, A_k{:}\,T_k\ \underline{\text{End}}. \end{aligned}$$

$\mathcal{D}_{\mathcal{S}}$ contains at most one such declaration for each class $C \in \mathcal{C}_{\mathcal{S}}$. ■

```
Class Course type-is
  Record
    enrolls: Set-of Student,
    taughtby: Teacher
  End

Class Teacher type-is
  Union Professor, GradStud
  End

Class GradStud is-a Student type-is
  Record
    degree: String
  End
```

Figure 3.4: An object-oriented schema

To avoid confusion we will refer to the classes and attributes in the object-oriented model as "oo-classes" and "oo-attributes" respectively, while we use the terms "class" and "attribute" with the usual meaning.

Example 3.4.2 Figure 3.4 shows the object-oriented schema corresponding to a fragment of the KEE knowledge base of Figure 3.1. ∎

Each class declarations imposes constraints on the instances of the class it refers to. The is-a part of a class declaration allows to specify inclusion between the sets of instances of the involved oo-classes, while the type-is part specifies through a type expression the structure assigned to the objects that are instances of the oo-class.

3.4.2 Semantics of an Object-Oriented Model

The meaning of an object-oriented schema is given by specifying the characteristics of an instance of the schema. The definition of instance makes use of the notions of object identifiers and values. Given an object-oriented schema $\mathcal{S}$ and a finite set $\mathcal{O}_{id}$ of *object identifiers* denoting real world objects, the set $\mathcal{V}$ of *values* over $\mathcal{S}$ and $\mathcal{O}_{id}$ is inductively defined as follows:

- $\mathcal{O}_{id} \subseteq \mathcal{V}$.
- If $v_1, \ldots, v_k \in \mathcal{V}$ then $\{v_1, \ldots, v_k\} \in \mathcal{V}$.
- If $v_1, \ldots, v_k \in \mathcal{V}$ then $[A_1 \colon v_1, \ldots, A_k \colon v_k] \in \mathcal{V}$.
- Nothing else is in $\mathcal{V}$.

A *database instance* $\mathcal{I}$ of a schema $\mathcal{S}$ is constituted by a *finite* set $\mathcal{O}_{id}^{\mathcal{I}}$ of object identifiers[3], a mapping $\pi^{\mathcal{I}}$ assigning to each oo-class a subset of $\mathcal{O}_{id}^{\mathcal{I}}$, and a mapping $\rho^{\mathcal{I}}$ assigning a value in $\mathcal{V}$ to each object in $\mathcal{O}_{id}^{\mathcal{I}}$. The interpretation of type expressions in $\mathcal{I}$ is defined through an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each type expression a subset of $\mathcal{V}$ such that the following conditions are satisfied:

$$
\begin{aligned}
C^{\mathcal{I}} &= \pi^{\mathcal{I}}(C) \\
(\underline{\text{Union}}\ T_1, \ldots, T_k\ \underline{\text{End}})^{\mathcal{I}} &= T_1^{\mathcal{I}} \cup \cdots \cup T_k^{\mathcal{I}} \\
(\underline{\text{Set-of}}\ T)^{\mathcal{I}} &= \{\{v_1, \ldots, v_k\} \mid k \geq 0, \\
&\qquad v_i \in T^{\mathcal{I}}, \text{ for } i \in \{1, \ldots, k\}\} \\
(\underline{\text{Record}}\ A_1 \colon T_1, \ldots, A_k \colon T_k\ \underline{\text{End}})^{\mathcal{I}} &= \{[A_1 \colon v_1, \ldots, A_h \colon v_h] \mid h \geq k, \\
&\qquad v_i \in T_i^{\mathcal{I}}, \text{ for } i \in \{1, \ldots, k\}, \\
&\qquad v_j \in \mathcal{V}, \text{ for } j \in \{k+1, \ldots, h\}\}.
\end{aligned}
$$

[3]The object identifiers in $\mathcal{O}_{id}^{\mathcal{I}}$ are unstructured and therefore different from polymorphic objects as defined in Section 2.1.8.

Definition 3.4.3 Let $\mathcal{S} := (\mathcal{C}_\mathcal{S}, \mathcal{A}_\mathcal{S}, \mathcal{D}_\mathcal{S})$ be an object-oriented schema. A database instance $\mathcal{I}$ is said to be *legal* with respect to $\mathcal{S}$ if for each declaration

$$\underline{\text{Class}}\ C\ \underline{\text{is-a}}\ C_1, \ldots, C_n\ \underline{\text{type-is}}\ T$$

in $\mathcal{D}$, it holds that $C^\mathcal{I} \subseteq C_i^\mathcal{I}$ for each $i \in \{1, \ldots, n\}$, and that $\rho^\mathcal{I}(C^\mathcal{I}) \subseteq T^\mathcal{I}$. ∎

3.4.3 Relationship between Object-Oriented Schemata and $\mathcal{L}$-Schemata

We establish now a relationship between $\mathcal{LUF}$ and the object-oriented language presented above. This is done by providing a mapping from object-oriented schemata into primitive $\mathcal{LUF}$-schemata. Since the interpretation domain for $\mathcal{LUF}$-schemata consists of atomic objects, whereas each instance of an object-oriented schema is assigned a possibly structured value (see the definition of $\mathcal{V}$), we need to explicitly represent some of the notions that underlie the object-oriented language. In particular, while there is a correspondence between classes (in $\mathcal{LUF}$) and oo-classes, one must explicitly account for the type structure of each oo-class. This can be accomplished by introducing in $\mathcal{LUF}$ classes `AbstractClass`, to represent the oo-classes, and `RecType` and `SetType` to represent the corresponding types. The associations between oo-classes and types induced by the oo-class declarations, as well as the basic characteristics of types, are modeled by means of $\mathcal{LUF}$-attributes: the (functional) attribute `value` models the association between oo-classes and types, and the attribute `member` is used for specifying the type of the elements of a set. Moreover, the $\mathcal{LUF}$-classes representing types are assumed to be mutually disjoint, and disjoint from the classes representing oo-classes. These constraints are expressed by adequate inclusion assertions that will be part of the $\mathcal{LUF}$-schema we are going to define.

We first define the function ψ that maps each type expression into an $\mathcal{LUF}$-class expression as follows:

- Every oo-class C is mapped into a class name $\psi(C)$.
- Every type expression $\underline{\text{Union}}\ T_1, \ldots, T_k\ \underline{\text{End}}$ is mapped into $\psi(T_1) \sqcup \cdots \sqcup \psi(T_k)$.
- Every type expression $\underline{\text{Set-of}}\ T$ is mapped into $\texttt{SetType} \sqcap \forall \texttt{member}.\psi(T)$.
- Every oo-attribute A is mapped into an attribute name $\psi(A)$, and every type expression $\underline{\text{Record}}\ A_1\!:\!T_1, \ldots, A_k\!:\!T_k\ \underline{\text{End}}$ is mapped into
$$\texttt{RecType} \sqcap \forall \psi(A_1).\psi(T_1) \sqcap \exists^{=1}\psi(A_1) \sqcap \cdots \sqcap \forall \psi(A_k).\psi(T_k) \sqcap \exists^{=1}\psi(A_k).$$

Using ψ we define the $\mathcal{LUF}$-schema corresponding to a schema in the object-oriented model we have defined.

Definition 3.4.4 The primitive $\mathcal{LUF}$-schema $\psi(\mathcal{S}) := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ corresponding to the object-oriented schema $\mathcal{S} := (\mathcal{C}_\mathcal{S}, \mathcal{A}_\mathcal{S}, \mathcal{D}_\mathcal{S})$ is obtained as follows:

- $\mathcal{C} := \{\texttt{AbstractClass}, \texttt{RecType}, \texttt{SetType}\} \cup \{\psi(C) \mid C \in \mathcal{C}_\mathcal{S}\}$.
- $\mathcal{A} := \{\texttt{value}, \texttt{member}\} \cup \{\psi(A) \mid A \in \mathcal{A}_\mathcal{S}\}$.
- $\mathcal{T}$ consists of the following assertions:

$$\begin{aligned} \texttt{SetType} &\dot{\preceq} \neg\texttt{AbstractClass} \sqcap \neg\texttt{RecType} \\ \texttt{RecType} &\dot{\preceq} \neg\texttt{AbstractClass} \end{aligned}$$

and for each class declaration

$$\underline{\text{Class}}\ C\ \underline{\text{is-a}}\ C_1, \ldots, C_n\ \underline{\text{type-is}}\ T$$

in $\mathcal{S}$, an inclusion assertion

$$\begin{aligned} \psi(C) \dot{\preceq}\ & \texttt{AbstractClass} \sqcap \\ & \psi(C_1) \sqcap \cdots \sqcap \psi(C_n) \sqcap \\ & \forall \texttt{value}.\psi(T) \sqcap \exists^{=1}\texttt{value}. \end{aligned}$$

∎

From the above translation we can observe that inverse attributes are not necessary for the formalization of object-oriented data models. Indeed, the possibility of referring to the inverse of an attribute is generally ruled out in such models. However, recent papers (see for example [6, 43]) point out that this strongly limits the expressive power of the data model. Note also that the use of number restrictions is limited to the value 1, which corresponds to existence constraints and functionality, whereas union is used in a more general form than for example in the KEE system.

We illustrate the translation on an example.

Example 3.4.2 (cont.) The primitive $\mathcal{LUF}$-schema that corresponds to the object-oriented schema of Figure 3.4 is $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$, where

$\mathcal{C} := \{$`AbstractClass`, `RecType`, `SetType`, `String`,
`Course`, `Teacher`, `Professor`, `Student`, `GradStud`$\}$,

$\mathcal{A} := \{$`value`, `member`, `enrolls`, `taughtby`, `degree`$\}$,

and the set $\mathcal{T}$ of assertions consists of:

$$
\begin{array}{rcl}
\texttt{Course} & \dot{\preceq} & \texttt{AbstractClass} \sqcap \exists^{=1}\texttt{value} \sqcap \\
 & & \forall\texttt{value.}(\texttt{RecType} \sqcap \exists^{=1}\texttt{enrolls} \sqcap \exists^{=1}\texttt{taughtby} \sqcap \\
 & & \quad \forall\texttt{enrolls.}(\texttt{SetType} \sqcap \forall\texttt{member.Student}) \sqcap \\
 & & \quad \forall\texttt{taughtby.Teacher}) \\
\texttt{Teacher} & \dot{\preceq} & \texttt{AbstractClass} \sqcap \exists^{=1}\texttt{value} \sqcap \forall\texttt{value.}(\texttt{GradStud} \sqcup \texttt{Professor}) \\
\texttt{GradStud} & \dot{\preceq} & \texttt{AbstractClass} \sqcap \texttt{Student} \sqcap \exists^{=1}\texttt{value} \sqcap \\
 & & \forall\texttt{value.}(\texttt{RecType} \sqcap \exists^{=1}\texttt{degree} \sqcap \forall\texttt{degree.String}) \\
\texttt{SetType} & \dot{\preceq} & \neg\texttt{AbstractClass} \sqcap \neg\texttt{RecType} \\
\texttt{RecType} & \dot{\preceq} & \neg\texttt{AbstractClass}
\end{array}
$$

■

The effectiveness of the translation ψ is sanctioned by the following proposition.

Proposition 3.4.5 *For every object-oriented schema $\mathcal{S}$, there exist two mappings $\alpha_{\mathcal{S}}$ and $\beta_{\mathcal{S}}$ between instances of $\mathcal{S}$ and finite interpretations of its translation $\psi(\mathcal{S})$, such that:*

1. *For each legal instance $\mathcal{I}$ of $\mathcal{S}$, $\alpha_{\mathcal{S}}(\mathcal{I})$ is a finite model of $\psi(\mathcal{S})$, and for each type T, $T^{\mathcal{I}} \neq \emptyset$ if and only if $(\psi(T))^{\alpha_{\mathcal{S}}(\mathcal{I})} \neq \emptyset$.*
2. *For each finite model $\mathcal{M}$ of $\psi(\mathcal{S})$, $\beta_{\mathcal{S}}(\mathcal{M})$ is a legal instance of $\mathcal{S}$, and for each oo-class C, $(\psi(C))^{\mathcal{M}} \neq \emptyset$ if and only if $C^{\beta_{\mathcal{S}}(\mathcal{M})} \neq \emptyset$.*

Proof. The proof can be carried out following the lines of the proof of Proposition 3.3.8. □

The basic reasoning services considered in object-oriented databases are subtyping (check whether a type denotes a subset of another type in every legal instance) and type consistency (check whether a type is consistent in a legal instance). Based on Proposition 3.4.5, we can show that these forms of reasoning are fully captured by finite class consistency and finite class subsumption in primitive $\mathcal{LUF}$-schemata.

Theorem 3.4.6 *Let $\mathcal{S}$ be an object-oriented schema, T, T' two type expressions in $\mathcal{S}$, and $\psi(\mathcal{S})$ the translation of $\mathcal{S}$. Then the following holds:*

1. *T is consistent in $\mathcal{S}$ if and only if $\psi(\mathcal{S}) \not\models_f \psi(T) \equiv \bot$.*
2. *T is a subtype of T' in $\mathcal{S}$ if and only if $\psi(\mathcal{S}) \models_f \psi(T) \preceq \psi(T')$.*

Proof. The proof is analogous to the proof of Theorem 3.3.9, but it makes use of Proposition 3.4.5 instead of Proposition 3.3.8. □

Chapter 4

Intrinsic Complexity of Reasoning

While the complexity of reasoning on class expressions has been thoroughly investigated in the literature, deduction on schemata is not so well understood and far from being settled in all relevant cases. In this chapter we discuss the intrinsic complexity of reasoning on $\mathcal{L}$-schemata, and we concentrate on the case of primitive schemata. In particular, Section 4.2 gives lower bounds for the complexity of reasoning on both acyclic and general schemata in restricted languages, and the results are proved by exploiting direct reductions from well-known problems that are complete for the complexity classes **coNP**and **PSPACE**, respectively. We show that already reasoning on acyclic primitive $\mathcal{L}_0$-schemata is **coNP**-hard by exploiting a reduction from validity of propositional formulae in disjunctive normal form. It is interesting to notice that when we add either the possibility to express disjunction or the possibility to use cycles in the schema, class consistency becomes even **PSPACE**-hard. These results are established by exploiting two different reductions from validity of quantified boolean formulae. In Section 4.3 we introduce a general technique by which we can eliminate the constructor for qualified existential quantification from a schema expressed in $\mathcal{LC}$ or one of its sub-languages without influencing class consistency. This technique is a fruitful source of complexity results, the most relevant being **EXPTIME**-hardness of class consistency in general primitive $\mathcal{LU}$-schemata. In fact, it is surprising to notice that even when restricting the attention to primitive schemata, in which only sufficient conditions can be stated, the use of disjunctions together with cyclic assertions is sufficient to make reasoning as hard as in the most expressive decidable schema definition languages we consider in this thesis. Finally, Section 4.4 shows that there are different ways the constructors of $\mathcal{LT}$ may interact and lead to undecidability of class consistency. Section 4.1 some introductory notions about complexity classes.

4.1 Complexity Classes

We use standard notions from complexity theory as presented for example in [82, 124]. In particular, we will speak about the complexity classes **PTIME**, **NP**, **PSPACE**, and **EXPTIME**, whose definitions we briefly recall here.

The classes **PTIME**, **PSPACE**, and **EXPTIME** are defined in terms of the resources needed by a *deterministic* Turing Machine (TM) to solve a specific *decision problem*. A decision problem is characterized by the set of its *positive instances*, for which we assume some standard encoding as strings of symbols of the alphabet of the TM. The *size* of an instance X of a problem, denoted with $|X|$, is the length of its encoding. In this setting "time" is the number of transitions performed by the TM, and "space" is the number of tape positions effectively used by the TM. The TM *solves* the decision problem if, given as input a string encoding an instance of the problem, it decides whether such instance is positive. The class **PTIME** (resp. **PSPACE**, **EXPTIME**) contains the problems that can be solved by a deterministic TM in polynomial time (resp. polynomial space, exponential time) in the size of the input. **NP** contains the problems that can be solved by a *nondeterministic* TM in polynomial time.

The *complement* of a problem P is the set of instances that are not positive instances of P. Given a class $\mathcal{C}$, the class **co**$\mathcal{C}$ is the set of problems that are the complement of a problem in $\mathcal{C}$.

4.1.1 Complete Problems for Complexity Classes

Given a class $\mathcal{C}$, a problem P_1 is said to be $\mathcal{C}$-hard (w.r.t. the polynomial reduction) if for every problem P_2 in $\mathcal{C}$, there is a polynomial reduction from P_2 to P_1. If a $\mathcal{C}$-hard problem P_1 is in $\mathcal{C}$ then P_1 is said to be $\mathcal{C}$-complete.

A typical **NP**-complete problem is deciding the satisfiability of propositional formulae in conjunctive normal form. A typical **coNP**-complete problem is deciding the validity of propositional formulae in disjunctive normal form (DNF for short) [82, page 261]. We briefly recall the definition of the latter problem, which is the one we are going to use.

Definition 4.1.1 A *literal* is a nonzero integer. A *clause* is a nonempty finite set c of literals such that $l \in c$ implies $-l \notin c$. A clause c is *over* a set of positive integers P if for each $l \in c$, $|l| \in P$. A *formula in DNF* (over P) is a finite set of clauses (over P). A P-assignment is a mapping

$$P \to \{\mathbf{t}, \mathbf{f}\}.$$

A P-assignment α *satisfies* a literal l if $\alpha(l) = \mathbf{t}$ if l is positive and $\alpha(-l) = \mathbf{f}$ if l is negative. It *satisfies* a clause if it satisfies all literals in the clause[1], and it *satisfies* a formula in DNF if it satisfies at least one clause in the formula.[2]. A DNF formula over P is *valid* if all P-assignments satisfy it. ∎

A typical **PSPACE**-complete problem is deciding the validity of quantified boolean formulae [82, page 172], which generalizes validity of DNF formulae and which we briefly recall here.

Definition 4.1.2 A *prefix* from m to n, where m and n are positive integers such that $m \leq n$, is a sequence

$$(Q_n n)(Q_{n-1} n-1) \cdots (Q_m m),$$

where each Q_i is either "$\forall$" or "$\exists$". A *quantified boolean formula* is a pair $Q.f$, where, for some n, Q is a prefix from 1 to n and f is a formula in DNF over $\{1, \ldots, n\}$. Let Q be a prefix from m to n. A Q-assignment is an $\{m, m+1, \ldots, n\}$-assignment in the sense of Definition 4.1.1. A set $\mathbf{A}$ of Q-assignments is *canonical for* Q if it satisfies the following conditions:

1. $\mathbf{A}$ is nonempty.
2. If $Q = (\exists n)Q'$, then all assignments of $\mathbf{A}$ agree on n and, if Q' is nonempty, $\{\alpha_{|\{m,..,n-1\}} \mid \alpha \in \mathbf{A}\}$ is canonical for Q'.
3. If $Q = (\forall n)Q'$, then
   (a) $\mathbf{A}$ contains an assignment that satisfies n and, if Q' is nonempty, then $\{\alpha_{|\{m,..,n-1\}} \mid \alpha \in \mathbf{A}$ and $\alpha(n) = \mathbf{t}\}$ is canonical for Q'.
   (b) $\mathbf{A}$ contains an assignment that satisfies $-n$ and, if Q' is nonempty, then $\{\alpha_{|\{m,..,n-1\}} \mid \alpha \in \mathbf{A}$ and $\alpha(n) = \mathbf{f}\}$ is canonical for Q'.

A quantified boolean formula $Q.f$ is *valid* if there exists a set $\mathbf{A}$ of Q-assignments that is canonical for Q and such that every assignment in $\mathbf{A}$ satisfies at least one clause of f. ∎

4.2 Lower Bounds by Direct Reductions

In this section we provide direct reductions that allow us to establish various hardness results for reasoning on primitive schemata. In particular, we show that even for the simplest schema language we consider in this thesis, namely $\mathcal{L}_0$, reasoning is already computationally hard. We prove a **coNP**-hardness result for class consistency in acyclic primitive schemata and sketch how the proof can be extended to show **PSPACE**-hardness of class consistency in acyclic $\mathcal{LU}$-schemata. In order to show that the same lower bound holds also if we admit cycles in primitive $\mathcal{L}_0$-schemata we exploit the capability of a cyclic $\mathcal{L}_0$-schema to simulate an m-bit binary counter.

[1] This means that we view the clause as a conjunction of literals.
[2] This means that we view a formula as a disjunction of clauses.

4.2.1 Preliminaries

We first discuss some simple properties of $\mathcal{L}_0$-schemata that will be used in the rest of the section. Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be a primitive $\mathcal{L}_0$-schema, $\{A_1, \ldots, A_n\} \subseteq \mathcal{A}$ a set of attributes, and $\{C_0, \ldots, C_n\} \subseteq \mathcal{C}$ a set of classes of $\mathcal{S}$. We say that a sequence $K := C_0 C_1 \cdots C_n$ of classes is an $(A_1 \cdots A_n)$-*chain in* $\mathcal{S}$ (of length n), if $\mathcal{T}$ contains the following assertions:

$$C_{j-1} \;\dot{\preceq}\; \forall A_j.C_j, \qquad \text{for } j \in \{1, \ldots, n\}.$$

An $(A_1 \cdots A_n)$-chain where $A_1 = A_2 = \cdots = A_n =: A$ is simply an A-chain.

Let $\{C_{ij} \mid i \in \{1, \ldots, m\}, j \in \{0, \ldots, n\}\} \subseteq \mathcal{C}$, and $K_i := C_{i0} C_{i1} \cdots C_{in}$, $i \in \{1, \ldots, m\}$. We say that a set $H := \{K_1, \ldots, K_m\}$ of $(A_1 \cdots A_n)$-chains in $\mathcal{S}$ is *active in* $\mathcal{S}$, if for each $j \in \{1, \ldots, n\}$ there is an $i \in \{1, \ldots, m\}$ such that $\mathcal{T}$ contains the assertion $C_{i(j-1)} \;\dot{\preceq}\; \exists A_j$.

Intuitively, if a schema contains an active set H of chains, then in every model of the schema each object that is an instance of all initial classes of the chains in H requires the existence of a sequence of objects that are connected through the attributes of the chains and are instances of successive classes in the chains. This is formalized in the following lemma which establishes properties of classes involved in a set of active $(A_1 \cdots A_n)$-chains.

Lemma 4.2.1 *Let $K_i := C_{i0} C_{i1} \cdots C_{in}$, for $i \in \{1, \ldots, m\}$, and let $H := \{K_1, \ldots, K_m\}$ be an active set of $(A_1 \cdots A_n)$-chains in $\mathcal{S}$. Let further $\mathcal{I}$ be a model of $\mathcal{S}$. If there is an object $o \in \Delta^{\mathcal{I}}$ such that $o \in C_{i0}^{\mathcal{I}}$, for $i \in \{1, \ldots, m\}$, then for each $j \in \{1, \ldots, n\}$ there is an object $o_j \in \Delta^{\mathcal{I}}$ such that $o_j \in C_{ij}^{\mathcal{I}}$, for $i \in \{1, \ldots, m\}$.*

Proof. The proof is by induction on n. The base case for $n = 0$ is trivial. For the induction step, assume that H is an active set of $(A_1 \cdots A_{n+1})$-chains in $\mathcal{S}$ and that we have already shown that for each $j \in \{0, \ldots, n\}$ there is an object $o_j \in \Delta^{\mathcal{I}}$ such that $o_j \in C_{ij}^{\mathcal{I}}$, for $i \in \{1, \ldots, m\}$. Since H is active there is an i_0, $1 \leq i_0 \leq m$ such that $\mathcal{T}$ contains the assertion $C_{i_0 n} \;\dot{\preceq}\; \exists A_{n+1}$. $\mathcal{I}$ is a model of $\mathcal{S}$, and since $o_n \in C_{i_0 n}^{\mathcal{I}}$, there is an object o_{n+1} such that $(o_n, o_{n+1}) \in A_{n+1}^{\mathcal{I}}$. Since H is a set of $(A_1 \cdots A_{n+1})$-chains, $\mathcal{T}$ contains assertions $C_{in} \;\dot{\preceq}\; \forall A_{n+1}.C_{i(n+1)}$, for $i \in \{1, \ldots, m\}$. Therefore, for all $i \in \{1, \ldots, m\}$, since $o_n \in C_{in}^{\mathcal{I}}$ we have that $o_{n+1} \in C_{i(n+1)}^{\mathcal{I}}$. □

The next lemma can be seen as the converse of the one above. It states that if a set of chains is not active then there is a model in which an object populates all initial classes, but all final classes have an empty extension. Intuitively we can say that the generation of objects along the set of chains can be blocked at some point.

Lemma 4.2.2 *Let $n \geq 1$, $K_i := C_{i0} C_{i1} \cdots C_{in}$, for $i \in \{1, \ldots, m\}$, and let $\mathcal{S}$ be a schema consisting precisely of the set $H := \{K_1, \ldots, K_m\}$ of $(A_1 \cdots A_n)$-chains. If H is not active in $\mathcal{S}$ then there is a model $\mathcal{I}$ of $\mathcal{S}$ such that the following holds:*

- *There is an object $o \in \Delta^{\mathcal{I}}$ such that $o \in C_{i0}^{\mathcal{I}}$ for $i \in \{1, \ldots, m\}$.*
- *$C_{in}^{\mathcal{I}} = \emptyset$ for $i \in \{1, \ldots, m\}$.*

Proof. Since H is not active in $\mathcal{S}$, there is a $j_0 \in \{1, \ldots, n\}$ such that for no $i \in \{1, \ldots, m\}$, $\mathcal{T}$ contains the assertion $C_{i(j_0-1)} \;\dot{\preceq}\; \exists A_{j_0}$. It is easy to see that we obtain a model $\mathcal{I}$ of S by setting:

- $\Delta^{\mathcal{I}} := o_0, \ldots, o_{j_0-1}$.
- $C_{ij}^{\mathcal{I}} := \{o_j\}$, for $i \in \{1, \ldots, n\}$, $j \in \{0, \ldots, j_0 - 1\}$,
  $C_{ij}^{\mathcal{I}} := \emptyset$, for $i \in \{1, \ldots, n\}$, $j \in \{j_0, \ldots, m\}$.
- $A_j^{\mathcal{I}} := \{(o_{j-1}, o_j)\}$, for $j \in \{1, \ldots, j_0 - 1\}$,
  $A_j^{\mathcal{I}} := \emptyset$, for $j \in \{j_0, \ldots, n\}$.

□

In the rest of the section we are going to reduce decision problems that are hard for a certain complexity class to class consistency in suitable schemata. An instance of the decision problem is encoded by means of a set of chains whose activation depends on whether the given instance represents a positive instance of the problem. Lemmata 4.2.1 and 4.2.2 are used to relate the activation of a set of chains to the inconsistency of a class in the schema. In fact, the reductions we give show that the complexity of reasoning on $\mathcal{L}_0$-schemata lies precisely in the detection of sets of active chains.

4.2.2 Acyclic Primitive $\mathcal{L}_0$-Schemata

$\mathcal{L}_0$ is the simplest language we have considered and it is subsumed by almost all representation formalisms used both in Knowledge Representation and in Databases. Nevertheless reasoning on $\mathcal{L}_0$-schemata is already hard, as we are going to show, even when we consider their simplest variant containing only primitive class specifications and no cycles.

We show **coNP**-hardness of class consistency in acyclic primitive $\mathcal{L}_0$-schemata by reducing to it the problem of verifying the validity of a propositional formula f in DNF. To this end we exploit the encoding of f in a set of A-chains. The following construction is directly derived from a construction in [92], where **coNP**-hardness for class consistency in general primitive $\mathcal{LI}$-schemata is shown.

In the following, let $f := \{c_1, \ldots, c_n\}$ be a formula in DNF over $P := \{1, \ldots, m\}$. We construct an acyclic primitive $\mathcal{L}_0$-schema $\mathcal{S}_f$ such that a distinguished class of $\mathcal{S}_f$ is consistent if and only if f is valid. $\mathcal{S}_f$ consists of two parts $\mathcal{S}_P$ (which depends only on P) and $\mathcal{S}_c$ (which encodes the the clauses of f). We first define separately the two parts and then integrate them in a single schema.

Construction 4.2.3 The schema $\mathcal{S}_P := (\mathcal{C}_P, \mathcal{A}_P, \mathcal{T}_P)$ is defined as follows:
The set $\mathcal{C}_P$ of classes of $\mathcal{S}_P$ contains the following elements:

- $m+1$ classes $D_0, \ldots, D_m$.
- $m{\cdot}(m+1)$ classes E^+_{ij}, E^-_{ij}, for $i \in \{1, \ldots, m\}$, $j \in \{0, \ldots, i-1\}$.

The set of attributes of $\mathcal{S}_P$ is given by $\mathcal{A}_P := \{A^+, A^-\}$.
The set $\mathcal{T}_P$ of assertions of $\mathcal{S}_P$ contains the following elements:

- For each $i \in \{1, \ldots, m\}$, the assertion

$$D_i \;\dot{\preceq}\; \forall A^+.D_{i-1} \sqcap \exists A^+ \sqcap \forall A^-.D_{i-1} \sqcap \exists A^-. \tag{4.1}$$

- For each $i \in \{1, \ldots, m\}$, the assertion

$$D_i \;\dot{\preceq}\; \forall A^+.E^+_{i(i-1)} \sqcap \forall A^-.E^-_{i(i-1)}. \tag{4.2}$$

- For each $i \in \{2, \ldots, m\}$, $j \in \{1, \ldots, i-1\}$, the assertions

$$\begin{array}{lll} E^+_{ij} & \dot{\preceq} & \forall A^+.E^+_{i(j-1)} \sqcap \forall A^-.E^+_{i(j-1)} \\ E^-_{ij} & \dot{\preceq} & \forall A^+.E^-_{i(j-1)} \sqcap \forall A^-.E^-_{i(j-1)}. \end{array} \tag{4.3}$$

- For each $i \in \{1, \ldots, m\}$, $j \in \{0, \ldots, i-1\}$, the assertion

$$E^+_{ij} \;\dot{\preceq}\; \neg E^-_{ij}. \tag{4.4}$$

■

The interesting property of this schema is that in every model $\mathcal{I}$ of $\mathcal{S}_P$ with $D_m^{\mathcal{I}} \neq \emptyset$, for every possible P-assignment α there is an object $o_\alpha \in D_0^{\mathcal{I}}$ *corresponding to* α, as specified by the following lemma.

Lemma 4.2.4 *Let $\mathcal{I}$ be a model of the schema $\mathcal{S}_P$ obtained from P as specified in Construction 4.2.3, and let $D_m^{\mathcal{I}} \neq \emptyset$. Then for each P-assignment α there is an object $o_\alpha \in \Delta^{\mathcal{I}}$ such that for $i \in \{1, \ldots, m\}$ the following holds:*

- $o_\alpha \in D_0^{\mathcal{I}}$.
- $o_\alpha \in E_{i0}^{+\mathcal{I}}$, *if and only if* $\alpha(i) = \boldsymbol{t}$.
- $o_\alpha \in E_{i0}^{-\mathcal{I}}$, *if and only if* $\alpha(i) = \boldsymbol{f}$.

Proof. Let α be an arbitrary P-assignment and let $Seq_P := (A_m \cdots A_1)$ be the sequence of attributes such that $A_i = A^+$ if $\alpha(i) = \mathbf{t}$ and $A_i = A^-$ if $\alpha(i) = \mathbf{f}$. Consider the set $H_P := \{K_0, \ldots, K_m\}$ of Seq_P-chains in $\mathcal{S}_P$, where

$$\begin{array}{rcll} K_0 & := & D_m D_{m-1} \cdots D_0. & \\ K_i & := & \begin{cases} D_m \cdots D_i E^+_{i(i-1)} \cdots E^+_{i0}, & \text{if } \alpha(i) = \mathbf{t}, \\ D_m \cdots D_i E^-_{i(i-1)} \cdots E^-_{i0}, & \text{if } \alpha(i) = \mathbf{f}, \end{cases} & \text{for } i \in \{1, \ldots, m\}. \end{array}$$

By construction of $\mathcal{S}_P$, H_P is indeed a set of Seq_P-chains in $\mathcal{S}_P$ and moreover it is active. By Lemma 4.2.1 the claim holds for α. □

This property can be exploited for the reduction of validity as follows. The schema $\mathcal{S}_c$, which encodes the clauses of f, consists of a set of chains, which are attached to the final classes of schema $\mathcal{S}_P$. There is one chain for each literal, and the chains corresponding to literals i and $-i$ have E^+_{i0} and E^-_{i0}, respectively, as their initial classes. These chains are such that if there is a P-assignment α that does not satisfy f (i.e. in every clause there is at least one literal not satisfied by α) then a certain subset H_α of these chains becomes active. Additionally, we enforce a contradiction in the schema if we try to populate the final classes of these chains with a common instance. The assertions in S_P ensure that if we try to populate the initial class D_m we are indeed forced to generate for each α a common instance of all initial classes in the chains of H_α, and therefore generate a contradiction if α does not satisfy f. On the other hand, if f is valid, then there is no set of active chains that forces a contradiction and D_m is consistent in $\mathcal{S}_f$.

Construction 4.2.5 The schema $S_c := (\mathcal{C}_c, \mathcal{A}_c, \mathcal{T}_c)$ is defined as follows:
The set $\mathcal{C}_c$ of classes of $\mathcal{S}_c$ contains the following elements:

- $m{\cdot}(n+1)$ classes C^+_{ij}, for $i \in \{1, \ldots, m\}$, $j \in \{0, \ldots, n\}$.
- $m{\cdot}(n+1)$ classes C^-_{ij}, for $i \in \{1, \ldots, m\}$, $j \in \{0, \ldots, n\}$.

The set of attributes of $\mathcal{S}_c$ is given by $\mathcal{A}_c := \{A_c\}$.
The set $\mathcal{T}_c$ of assertions of $\mathcal{S}_c$ contains the following elements:

- For each $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ the assertions

$$\begin{array}{rcl} C^+_{i(j-1)} & \dot{\preceq} & \forall A_c.C^+_{ij} \\ C^-_{i(j-1)} & \dot{\preceq} & \forall A_c.C^-_{ij}. \end{array} \tag{4.5}$$

- For each $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$,
  - if $-i \in c_j$, the assertion
  $$C^+_{i(j-1)} \;\dot{\preceq}\; \exists A_c. \tag{4.6}$$
  - if $i \in c_j$, the assertion
  $$C^-_{i(j-1)} \;\dot{\preceq}\; \exists A_c. \tag{4.7}$$

■

We can now specify the set H_α of chains corresponding to a P-assignment α, as the set of m A_c-chains that start at $C^{*_i}_{i0}$, where "$*_i$" stands for "+" if $\alpha(i) = \mathbf{t}$, and for "−" if $\alpha(i) = \mathbf{f}$.

Definition 4.2.6 Let $\mathcal{S}_c$ be the schema obtained from a formula f in DNF as specified in Construction 4.2.5, and let the A_c-chains K_i^+ and K_i^- be given by $K_i^+ := C_{i0}^+ \cdots C_{in}^+$ and $K_i^- := C_{i0}^- \cdots C_{in}^-$. Then we say that K_i^+ *encodes* the clauses of f for literal i, and that K_i^- encodes the clauses of f for literal $-i$.

The set of A_c-chains *corresponding to* a P-assignment α is $H_\alpha := \{K_1, \ldots, K_m\}$, where for $i \in \{1, \ldots, m\}$.

$$K_i \;:=\; \begin{cases} K_i^+, & \text{if } \alpha(i) = \mathbf{t}. \\ K_i^-, & \text{if } \alpha(i) = \mathbf{f}. \end{cases}$$

■

Lemma 4.2.7 *Let $\mathcal{S}_c$ be the schema obtained from f as specified in Construction 4.2.5, and let α be a P-assignment. Then the set H_α of A-chains corresponding to α is active in $\mathcal{S}_c$ if and only if α does not satisfy f.*

Proof. "⇐" If α does not satisfy f, then for each clause c_j, $j \in \{1, \ldots, n\}$, there is a literal $l_j \in c_j$ that is not satisfied by α. If $l_j = i$, then $\alpha(i) = \mathbf{f}$ and by Definition 4.2.6 $K_i^- \in H_\alpha$, where $K_i^- := C_{i0}^- \cdots C_{in}^-$. Since $i \in c_j$, by Construction 4.2.5 $\mathcal{T}_c$ contains the assertion $C^-_{i(j-1)} \dot{\preceq} \exists A_c$. Similarly, if $l_j = -i$, then $\alpha(i) = \mathbf{t}$ and $K_i^+ \in H_\alpha$, where $K_i^+ := C_{i0}^+ \cdots C_{in}^+$, and since $-i \in c_j$, $\mathcal{T}_c$ contains the assertion $C^+_{i(j-1)} \dot{\preceq} \exists A_c$. Therefore H_α is active in $\mathcal{S}_c$.

"⇒" If α satisfies f, then there is a clause c_j such that for each literal $l \in c_j$, α satisfies l. If $l = i$ then $\alpha(i) = \mathbf{t}$ and $K_i^+ \in H_\alpha$. However $\mathcal{T}_c$ does not contain the assertion $C^+_{i(j-1)} \dot{\preceq} \exists A_c$. Similarly, if $l = -i$, then $\alpha(i) = \mathbf{f}$, $K_i^- \in H_\alpha$, but $\mathcal{T}_c$ does not contain the assertion $C^-_{i(j-1)} \dot{\preceq} \exists A_c$. Therefore $\mathcal{T}_c$ does not contain any assertion of the form $C^{*_i}_{i(j-1)} \dot{\preceq} \exists A_c$ for any $i \in \{1, \ldots, m\}, *_i \in \{+, -\}$, and H_α is not active in $\mathcal{S}_c$. □

By connecting the chains in $\mathcal{S}_c$ to the end of the chains in $\mathcal{S}_P$ and adding an assertion that forces a contradiction if a certain class is forced to be populated, we obtain the schema $\mathcal{S}_f$ that realizes the desired reduction.

Construction 4.2.8 The schema $\mathcal{S}_f := (\mathcal{C}_f, \mathcal{A}_f, \mathcal{T}_f)$ corresponding to formula f is defined as follows:

- $\mathcal{C}_f := \mathcal{C}_P \cup \mathcal{C}_c \cup \{F\}$.
- $\mathcal{A}_f := \mathcal{A}_P \cup \mathcal{A}_c$.
- $\mathcal{T}_f := \mathcal{T}_P \cup \mathcal{T}_c \cup \mathcal{T}'$, where $\mathcal{T}'$ contains the following assertions:
  For each $i \in \{1, \ldots, m\}$, the assertions

$$\begin{array}{lll} E_{i0}^+ & \dot{\preceq} & \forall A_c.C_{i0}^+ \\ E_{i0}^- & \dot{\preceq} & \forall A_c.C_{i0}^- \end{array} \tag{4.8}$$

$$\begin{array}{lll} C_{in}^+ & \dot{\preceq} & \forall A_c.F \\ C_{in}^- & \dot{\preceq} & \forall A_c.F. \end{array} \tag{4.9}$$

  Finally, the assertions

$$\begin{array}{lll} E_{10}^+ & \dot{\preceq} & \exists A_c \\ E_{10}^- & \dot{\preceq} & \exists A_c \\ C_{1n}^+ & \dot{\preceq} & \exists A_c \\ C_{1n}^- & \dot{\preceq} & \exists A_c \end{array} \tag{4.10}$$

  and

$$F \;\dot{\preceq}\; \neg F. \tag{4.11}$$

■

Clearly, $\mathcal{S}_f$ is an acyclic primitive $\mathcal{L}_0$-schema, and moreover it is easy to see that $|\mathcal{S}_f|$ is quadratic in $|f|$.

The presence of assertion 4.11 forces a contradiction in the schema in the case where class F is forced to be populated. In the following theorem we show that this is the case if and only if the formula f is not valid.

Theorem 4.2.9 *Let $f := \{c_1, \ldots, c_n\}$ be a formula in DNF over $P := \{l_1, \ldots, l_m\}$ and $\mathcal{S}_f$ be the acyclic primitive $\mathcal{L}_0$-schema obtained from f as specified in Construction 4.2.8. Then D_m is consistent in $\mathcal{S}_f$ if and only if f is valid.*

Proof. "⇒" Let α be a P-assignment that does not satisfy f, and assume that there is a model $\mathcal{I}$ of $\mathcal{S}_f$ such that $D_m^{\mathcal{I}} \neq \emptyset$. By Lemma 4.2.4 there is an object $o_\alpha \in \Delta^{\mathcal{I}}$ such that $o_\alpha \in D_0^{\mathcal{I}}$ and for $i \in \{1, \ldots, m\}$ $o_\alpha \in E_i^{\mathcal{I}}$, where

$$E_i \;:=\; \begin{cases} E_{im}^{+}, & \text{if } \alpha(i) = \mathbf{t}. \\ E_{im}^{-}, & \text{if } \alpha(i) = \mathbf{f}. \end{cases}$$

By Lemma 4.2.7 the set $H_\alpha := \{K_1, \ldots, K_m\}$ of A-chains corresponding to α is active in $\mathcal{S}_c$ and therefore also in $\mathcal{S}_f$. It follows by assertions 4.8, 4.9, and 4.10 in $\mathcal{S}_f$, that also

$$H'_\alpha \;:=\; \{E_i K_i F \mid i \in \{1, \ldots, m\}\}$$

is an active set of A-chains in $\mathcal{S}_f$. By Lemma 4.2.1 there is an object $o' \in \Delta^{\mathcal{I}}$ such that $o' \in F$. But by assertion 4.11 this contradicts the fact that $\mathcal{I}$ is a model of $\mathcal{S}_f$.

"⇐" Let f be valid. We show how to construct a model of S_f. By Lemma 4.2.4 we can obtain a model $\mathcal{I}_P$ of $\mathcal{S}_P$ where for each P-assignment α, $\Delta^{\mathcal{I}_P}$ contains a distinct object $o_\alpha \in D_0^{\mathcal{I}_P}$, such that $o_\alpha \in E_{i0}^{+\mathcal{I}_P}$, if and only if $\alpha(i) = \mathbf{t}$, and $o_\alpha \in E_{i0}^{-\mathcal{I}_P}$, if and only if $\alpha(i) = \mathbf{f}$, for $i \in \{1, \ldots, m\}$. By Lemma 4.2.7 each set H_α of A-chains corresponding to a P-assignment α is not active, and by Lemma 4.2.2, there is a model $\mathcal{I}_\alpha$ of H_α in which the final classes of the chains of H_α all have an empty extension and in which there is a single object o'_α that is an instance of all initial classes of the chains in H_α. We construct an interpretation $\mathcal{I}$ of $\mathcal{S}_f$ as follows: As domain of $\mathcal{I}$ we take the union of $\Delta^{\mathcal{I}_p}$ and all $dom[\mathcal{I}_\alpha]$. As extensions of classes and attributes we take the union of the extensions in the separate models and add to the extension of A_c all pairs (o_α, o'_α). It is not difficult to see that $\mathcal{I}$ is a model of $\mathcal{S}_f$. □

As an immediate consequence of Theorem 4.2.9 we get the following lower bound for class consistency in acyclic primitive $\mathcal{L}_0$-schemata.

Corollary 4.2.10 *Class consistency in acyclic primitive $\mathcal{L}_0$-schemata is **coNP**-hard.* □

We observe that the schema $\mathcal{S}_f$ as given by Construction 4.2.8 contains some redundancy that was added for simplifying the proofs. In particular, the disjointness assertions 4.4 between each class E_{ij}^{+} and the corresponding class E_{ij}^{-} can be omitted. The resulting schema then contains only one assertion in which the symbol "¬" appears, namely 4.11, and **coNP**-hardness of class consistency holds already for this special case. In fact, negation is needed only to introduce a single inconsistent class in the schema. The proof shows that the complexity of reasoning on primitive $\mathcal{L}_0$-schemata lies in the interaction that occurs between different chains of attributes and in the detection of sets of chains that are active. The number of different sets of chains is in general exponential, and their activation may be spread over the whole schema. Therefore, in order to detect if there is such a set that is active and forces one to populate an inconsistent class, in the worst case all sets of chains have to be tested.

We would like to notice that the **coNP**-hardness result for class consistency in acyclic primitive $\mathcal{L}_0$-schemata has been independently proven in [37]. This paper contains also a nondeterministic algorithm that decides in polynomial time whether a class in an acyclic primitive $\mathcal{L}_0$-schema is inconsistent. This shows that the problem is **coNP**-complete.

Corollary 4.2.11 *Class consistency in acyclic primitive $\mathcal{L}_0$-schemata is **coNP**-complete.* □

The hardness proof in [37], however, exploits a **coNP**-hardness result for verifying consistency of $\mathcal{LE}$-expressions, while the reduction we propose is more direct. It gives also a better insight on the structure of the problem and on the reasons that lead to intractability. Moreover, as shown below, the reduction can be generalized to show that class consistency becomes **PSPACE**-hard if we add the possibility of expressing disjunction.

4.2.3 Acyclic Primitive $\mathcal{LU}$-Schemata

We adapt now the **coNP**-hardness proof for acyclic primitive $\mathcal{L}_0$-schemata to prove **PSPACE**-hardness for class consistency in acyclic primitive $\mathcal{LU}$-schemata.

We start by observing that validity of formulae in DNF is just a particular case of validity of quantified boolean formulae. If f is a formula in DNF over $P := \{1, \ldots, m\}$, then f is valid if and only if the quantified boolean formula $q_f := (\forall m)(\forall m-1)\cdots(\forall 1).f$ is valid. The schema $\mathcal{S}_P$ defined in Construction 4.2.3 is such that in any model $\mathcal{I}$ of $\mathcal{S}_P$ in which $D_m^{\mathcal{I}} \neq \emptyset$, for each P-assignment α there is an object $o_\alpha \in \Delta^{\mathcal{I}}$ corresponding to α. The fact that this holds for all P-assignments takes into account that in the prefix of q_f all quantifiers are equal to "$\forall$". In a generic quantified boolean formula $Q.f$ not all assignments have to be considered but only those that together constitute some set $\mathbf{A}$ of Q-assignments canonical for Q. Therefore, in order to reduce validity of $Q.f$ to class consistency, it is sufficient to use a schema $\mathcal{S}_Q$ which forces for each of its models only the existence of exactly those objects that correspond to the assignments in some $\mathbf{A}$. This can in fact be achieved through the introduction of disjunction.

Construction 4.2.12 Let $Q := (Q_m m)\cdots(Q_1 1)$ be a prefix from 1 to m. The schema $\mathcal{S}_Q := (\mathcal{C}_Q, \mathcal{A}_Q, \mathcal{T}_Q)$ is defined as follows:
The set $\mathcal{C}_Q$ of classes of $\mathcal{S}_Q$ contains the following elements:

- $m+1$ classes $D_0, \ldots, D_m$.
- For each $i \in \{1, \ldots, m\}$ such that $Q_i = \exists$, two classes D_{i-1}^+ and D_{i-1}^-.
- $m{\cdot}(m+1)$ classes E_{ij}^+, E_{ij}^-, for $i \in \{1, \ldots, m\}$, $j \in \{0, \ldots, i-1\}$.

The set of attributes of $\mathcal{S}_P$ is given by $\mathcal{A}_P := \{A^+, A^-\}$.
The set $\mathcal{T}_P$ of assertions of $\mathcal{S}_P$ contains the following elements:

- For each $i \in \{1, \ldots, m\}$, the assertion

$$D_i \;\dot{\preceq}\; \forall A^+.D_{i-1} \sqcap \forall A^-.D_{i-1}. \tag{4.12}$$

- For each $i \in \{1, \ldots, m\}$,
  - if $Q_i = \forall$, the assertion

$$D_i \;\dot{\preceq}\; \forall A^+.E_{ii-1}^+ \sqcap \exists A^+ \sqcap \forall A^-.E_{ii-1}^- \sqcap \exists A^-. \tag{4.13}$$

  - if $Q_i = \exists$, the assertions

$$\begin{array}{lcl} D_i & \dot{\preceq} & D_i^+ \sqcup D_i^- \\ D_i^+ & \dot{\preceq} & D_i \sqcap \forall A^+.E_{ii-1}^+ \sqcap \exists A^+ \\ D_i^- & \dot{\preceq} & D_i \sqcap \forall A^-.E_{ii-1}^- \sqcap \exists A^- \\ D_i^+ & \dot{\preceq} & \neg D_i^-. \end{array} \tag{4.14}$$

- For each $i \in \{2, \ldots, m\}$, $j \in \{1, \ldots, i-1\}$, the assertions

$$\begin{array}{lcl} E_{ij}^+ & \dot{\preceq} & \forall A^+.E_{i(j-1)}^+ \sqcap \forall A^-.E_{i(j-1)}^+ \\ E_{ij}^- & \dot{\preceq} & \forall A^+.E_{i(j-1)}^- \sqcap \forall A^-.E_{i(j-1)}^-. \end{array} \tag{4.15}$$

- For each $i \in \{1, \ldots, m\}$, $j \in \{0, \ldots, i-1\}$, the assertion

$$E_{ij}^+ \;\dot{\preceq}\; \neg E_{ij}^-. \tag{4.16}$$

■

Lemma 4.2.4 generalizes to this case as follows.

Lemma 4.2.13 *Let $\mathcal{S}_Q$ be the schema obtained from Q as specified in Construction 4.2.12. Then for every model $\mathcal{I}$ of $\mathcal{S}_Q$ with $D_m^{\mathcal{I}} \neq \emptyset$ there is a set $\boldsymbol{A}$ of Q-assignments canonical for Q, and for every set $\boldsymbol{A}$ of Q-assignments canonical for Q there is a model $\mathcal{I}$ of $\mathcal{S}_Q$ with $D_m^{\mathcal{I}} \neq \emptyset$ such that the following holds: For every Q-assignment $\alpha \in \boldsymbol{A}$ there is an object $o_\alpha \in \Delta^{\mathcal{I}}$ such that for $i \in \{1, \ldots, m\}$*

- $o_\alpha \in D_0^{\mathcal{I}}$.
- $o_\alpha \in E_{i0}^{+\mathcal{I}}$, *if and only if* $\alpha(i) = \boldsymbol{t}$.
- $o_\alpha \in E_{i0}^{-\mathcal{I}}$, *if and only if* $\alpha(i) = \boldsymbol{f}$.

Proof. The proof is by induction on the length of the prefix Q. In fact, for carrying on the induction we need to show that there is a distinct object $o_\alpha \in D_0^{\mathcal{I}}$ for each object $o \in D_m^{\mathcal{I}}$.

For the base case, if $Q = \forall 1$, then there is only one set $\mathbf{A}$ of Q-assignments canonical for Q. $\mathbf{A}$ consists of the two Q-assignments $\alpha_{\mathbf{t}}$ and $\alpha_{\mathbf{f}}$, with $\alpha_{\mathbf{t}}(1) = \mathbf{t}$ and $\alpha_{\mathbf{f}}(1) = \mathbf{f}$. $\mathcal{S}_Q$ admits a model $\mathcal{I}$ with $D_1^{\mathcal{I}} \neq \emptyset$, and by assertion 4.13 every such model contains two objects o^+, o^-, with $o^+ \in E_{10}^{+\mathcal{I}}$ and $o^- \in E_{10}^{-\mathcal{I}}$, i.e. o^+ corresponds to $\alpha_{\mathbf{t}}$ and o^- corresponds to $\alpha_{\mathbf{f}}$.

If $Q = \exists 1$, then there are two sets of Q-assignments canonical for Q, $\mathbf{A_t} := \{\alpha_{\mathbf{t}}\}$ and $\mathbf{A_f} := \{\alpha_{\mathbf{f}}\}$. $\mathcal{S}_Q$ admits two types of models $\mathcal{I}$ with $D_1^{\mathcal{I}} \neq \emptyset$, those with $D_1^{+\mathcal{I}} \neq \emptyset$ and those with $D_1^{-\mathcal{I}} \neq \emptyset$. By assertions 4.14, every such model $\mathcal{I}$ of the first type contains an object o^+, with $o^+ \in E_{10}^{+\mathcal{I}}$ and every such model $\mathcal{I}$ of the second type contains an object o^-, with $o^- \in E_{10}^{-\mathcal{I}}$.

For the induction step, assume that the claim holds for a prefix Q of length m. If $Q' := \forall(m+1)Q$, let $\mathcal{I}$ be a model of $\mathcal{S}_{Q'}$ and $o' \in D_{m+1}^{\mathcal{I}}$. By assertions 4.13 there are two objects $o^+ \in E_{(m+1)m}^{+\mathcal{I}}$ and $o^- \in E_{(m+1)m}^{-\mathcal{I}}$, and by assertions 4.12, $o^+, o^- \in D_m^{\mathcal{I}}$. By induction hypothesis, there are two sets $\mathbf{A}^+$ and $\mathbf{A}^-$ of Q-assignments canonical for Q such that for each $\alpha \in \mathbf{A}^+$ there is an object o_α^+ and for each $\alpha \in \mathbf{A}^-$ there is an object o_α^- satisfying the conditions of the claim. By assertions 4.15 (with $i = m+1$), $o_\alpha^+ \in E_{(m+1)0}^{+\mathcal{I}}$ and $o_\alpha^- \in E_{(m+1)0}^{-\mathcal{I}}$. The first direction of the claim follows by observing that we obtain a set $\mathbf{A}'$ of Q'-assignments canonical for Q' by taking the union of the assignments α in $\mathbf{A}^+$ extended by $\alpha(m+1) = \mathbf{t}$ and the assignments α in $\mathbf{A}^-$ extended by $\alpha(m+1) = \mathbf{f}$.

Conversely, let $\mathbf{A}'$ be a set of Q'-assignments canonical for Q'. By definition the sets of Q-assignments $\mathbf{A}^+ := \{\alpha_{|\{0,\ldots,m\}} \mid \alpha \in \mathbf{A}' \text{ and } \alpha(m+1) = \mathbf{t}\}$ and $\mathbf{A}^- := \{\alpha_{|\{0,\ldots,m\}} \mid \alpha \in \mathbf{A}' \text{ and } \alpha(m+1) = \mathbf{f}\}$ are canonical for Q. By induction hypothesis there are two models $\mathcal{I}_1$ and $\mathcal{I}_2$ of $\mathcal{S}_Q$ for which the following holds:

- $\mathcal{I}_1$ contains an object $o^+ \in D_m^{\mathcal{I}_1}$ and for each Q-assignment $\alpha \in \mathbf{A}^+$ an object o_α satisfying the conditions of the lemma.
- $\mathcal{I}_2$ contains an object $o^- \in D_m^{\mathcal{I}_2}$ and for each Q-assignment $\alpha \in \mathbf{A}^-$ an object o_α satisfying the conditions of the lemma.

We assume without loss of generality that $\Delta^{\mathcal{I}_1}$ and $\Delta^{\mathcal{I}_2}$ are disjoint and do not contain o'. It is easy to see that we obtain a model $\mathcal{I}$ of $\mathcal{S}_{Q'}$ satisfying the required conditions as follows:

- $\Delta^{\mathcal{I}} := \Delta^{\mathcal{I}_1} \cup \Delta^{\mathcal{I}_2} \cup \{o\}$.
- $D_{m+1}^{\mathcal{I}} := \{o'\}$,
  $D_i^{\mathcal{I}} := D_i^{\mathcal{I}_1} \cup D_i^{\mathcal{I}_1}$, for $i \in \{1, \ldots, m\}$,
  $E_{ij}^{*\mathcal{I}} := E_{ij}^{*\mathcal{I}_1} \cup E_{ij}^{*\mathcal{I}_1}$, for $i \in \{1, \ldots, m\}$, $j \in \{0, \ldots, i-1\}$, $* \in \{+, -\}$,
  $E_{(m+1)j}^{+\mathcal{I}} := D_j^{\mathcal{I}_1}$, $E_{(m+1)j}^{-\mathcal{I}} := D_j^{\mathcal{I}_1}$, for $j \in \{0, \ldots, m-1\}$.
- $A^{+\mathcal{I}} := A^{+\mathcal{I}_1} \cup A^{+\mathcal{I}_2} \cup \{(o', o^+)\}$,
  $A^{-\mathcal{I}} := A^{-\mathcal{I}_1} \cup A^{-\mathcal{I}_2} \cup \{(o', o^-)\}$.

If $Q' := \exists(m+1)Q$, let $\mathcal{I}$ be a model of $\mathcal{S}_{Q'}$ and $o' \in D_{m+1}^{\mathcal{I}}$. By assertions 4.14, either $o' \in D_{m+1}^{+\mathcal{I}}$ or $o' \in D_{m+1}^{-\mathcal{I}}$. We discuss the case where $o' \in D_{m+1}^{+\mathcal{I}}$, the other case being symmetric. Then there is an object o with $o \in E_{(m+1)m}^{+\mathcal{I}}$, and by assertions 4.12, $o \in D_m^{\mathcal{I}}$. By induction hypothesis, there is a set $\mathbf{A}$ of

Q-assignments canonical for Q such that for each $\alpha \in \mathbf{A}$ there is an object o_α satisfying the conditions of the claim. By assertions 4.15 (with $i = m+1$), $o_\alpha \in E^{+\mathcal{I}}_{(m+1)0}$. The first direction of the claim follows by observing that we obtain a set $\mathbf{A}'$ of Q'-assignments canonical for Q' by extending each assignment α in $\mathbf{A}^+$ by $\alpha(m+1) = \mathbf{t}$.

Conversely, let $\mathbf{A}'$ be a set of Q'-assignments canonical for Q'. By definition, all Q'-assignments agree on $m+1$. We discuss the case where for each Q'-assignment α, $\alpha(m+1) = \mathbf{t}$, the other case being symmetric. Then the set of Q-assignments $\mathbf{A} := \{\alpha_{|\{0,\ldots,m\}} \mid \alpha \in \mathbf{A}' \text{ and } \alpha(m+1) = \mathbf{t}\}$ is canonical for Q. By induction hypothesis there is a model $\mathcal{I}$ of $\mathcal{S}_Q$ containing an object $o \in D^{\mathcal{I}}_m$ and for each Q-assignment $\alpha \in \mathbf{A}$ an object o_α satisfying the conditions of the lemma. We assume without loss of generality that $o' \notin \Delta^{\mathcal{I}}$. It is easy to see that we obtain a model $\mathcal{I}'$ of $\mathcal{S}_{Q'}$ satisfying the required conditions as follows:

- $\Delta^{\mathcal{I}'} := \Delta^{\mathcal{I}} \cup \{o\}$.
- $D^{\mathcal{I}'}_{m+1} := \{o\}$,
  $D^{\mathcal{I}'}_i := D^{\mathcal{I}}_i$, for $i \in \{1,\ldots,m\}$,
  $E^{*\mathcal{I}'}_{ij} := E^{*\mathcal{I}}_{ij}$, for $i \in \{1,\ldots,m\}$, $j \in \{0,\ldots,i-1\}$, $* \in \{+,-\}$,
  $E^{+\mathcal{I}'}_{(m+1)j} := D^{\mathcal{I}}_j$, $E^{-\mathcal{I}'}_{(m+1)j} := \emptyset$, for $j \in \{0,\ldots,m-1\}$.
- $A^{+\mathcal{I}'} := A^{+\mathcal{I}} \cup \{(o', o)\}$,
  $A^{-\mathcal{I}'} := A^{-\mathcal{I}}$.

□

We can then construct a schema S_q corresponding to a quantified boolean formula $q := Q.f$, by composing the schema $\mathcal{S}_Q$ obtained using Construction 4.2.12 with the schema $\mathcal{S}_f$ obtained using Construction 4.2.5 as specified in Construction 4.2.8 where we use $\mathcal{S}_Q$ instead of $\mathcal{S}_P$. Applying Lemma 4.2.13 we can prove the following generalization of Theorem 4.2.9 to quantified boolean formulae.

Theorem 4.2.14 *Let $q := Q.f$ be a quantified boolean formula and $\mathcal{S}_q$ be the acyclic primitive $\mathcal{LU}$-schema obtained from f as specified above. Then D_m is consistent in $\mathcal{S}_q$ if and only if q is valid.*

Proof. The proof is similar to the proof of Theorem 4.2.9, using Lemma 4.2.13 instead of Lemma 4.2.4. □

As an immediate consequence of Theorem 4.2.14 we get the following lower bound for class consistency in acyclic primitive $\mathcal{LU}$-schemata.

Corollary 4.2.15 *Class consistency in acyclic primitive $\mathcal{LU}$-schemata is **PSPACE**-hard.* □

4.2.4 General Primitive $\mathcal{L}_0$-Schemata

We show now that if we admit cycles in a primitive $\mathcal{L}_0$-schema we obtain the same lower bound for class consistency as for acyclic $\mathcal{LU}$-schemata, namely **PSPACE**-hardness. The problem of deciding class consistency in primitive $\mathcal{L}_0$-schema is studied also in [37], where an algorithm to solve it is given that requires polynomial space in the size of the schema. The reduction we provide closes the complexity gap between the lower bound that holds already for acyclic schemata and the **PSPACE**-algorithm.

We propose again a reduction from validity of quantified boolean formulae. However, different from the case of class consistency in $\mathcal{LU}$-schemata, we do not make use of the technique developed for acyclic $\mathcal{L}_0$-schemata and based on the generation of objects that correspond to truth assignments. In fact, there seems to be no easy way to cope with the presence of existential quantifiers in the prefix without making use of disjunction. For this reason, for the current reduction we exploit the capability of general $\mathcal{L}_0$-schemata to simulate a binary counter, as shown by the following construction.

Construction 4.2.16 The schema $S_m := (\mathcal{C}_m, \mathcal{A}_m, \mathcal{T}_m)$ that simulates an m-bit counter is defined as follows:
The set $\mathcal{C}_m$ of classes of $\mathcal{S}_m$ is given by $2m$ classes $B^-_1, \ldots, B^-_m, B^+_1, \ldots, B^+_m$.
The set of attributes of $\mathcal{S}_m$ is given by $\mathcal{A}_m := \{A_1, \ldots, A_m\}$.

The set $\mathcal{T}_m$ of assertions of $\mathcal{S}_m$ contains for each $i \in \{1, \ldots, m\}$ the following assertions:

$$\begin{array}{lcl} B_i^- & \dot{\preceq} & \forall A_1.B_i^- \sqcap \forall A_2.B_i^- \sqcap \cdots \sqcap \forall A_{i-1}.B_i^- \\ B_i^- & \dot{\preceq} & \forall A_i.B_i^+ \sqcap \exists A_i \\ B_i^+ & \dot{\preceq} & \forall A_1.B_i^+ \sqcap \forall A_2.B_i^+ \sqcap \cdots \sqcap \forall A_{i-1}.B_i^+ \\ B_i^+ & \dot{\preceq} & \forall A_{i+1}.B_i^- \sqcap \forall A_{i+2}.B_i^- \sqcap \cdots \sqcap \forall A_m.B_i^- \\ B_i^- & \dot{\preceq} & \neg B_i^+. \end{array} \tag{4.17}$$

■

We have chosen assertions 4.17 in such a way that B_1^- and B_1^+ represent the least significant bit of the counter. When we are going to embed the encoding of a quantified boolean formula in the counter the least significant bit will correspond to the innermost quantifier in the prefix, which quantifies over the letter "1".

If in a model of this schema the classes $B_1^-, \ldots, B_m^-$ contain a common instance, then for each number u between 0 and $2^m - 1$ there is an object o_u which is a common instance of the classes that give the binary representation of u. This is formalized in the following lemma. We express the fact that the existence of an object corresponding to a certain number is enforced by saying that the count *proceeds* till that number.

Lemma 4.2.17 *Let $\mathcal{S}_m$ be the schema obtained as specified in Construction 4.2.16 and $\mathcal{I}$ a model of $\mathcal{S}_m$ containing an object o with $o \in B_i^{-\mathcal{I}}$, for $i \in \{1, \ldots, m\}$. Then there is a sequence of (not necessarily distinct) objects $o_0, o_1, \ldots, o_{2^m-1} \in \Delta^{\mathcal{I}}$ such that $o = o_0$ and for $u \in \{0, \ldots, 2^m - 1\}$, if the binary representation of u is $b_m b_{m-1} \cdots b_1$, i.e. $u = \sum_{i=1}^{m} 2^{i-1} \cdot b_i$, then:*

- *$o_u \in B_i^{-\mathcal{I}}$ if $b_i = 0$.*
- *$o_u \in B_i^{+\mathcal{I}}$ if $b_i = 1$.*
- *if $u \neq 2^m - 1$, $(o_u, o_{u+1}) \in A_{i_u}^{\mathcal{I}}$, where i_u is the least i such that $b_i = 0$.*

Proof. The proof is by induction on u.

For $u = 0$, by assumption $o_0 \in B_i^{-\mathcal{I}}$, for $i \in \{1, \ldots, m\}$, and by assertion $B_1^- \dot{\preceq} \forall A_1.B_1^+ \sqcap \exists A_1$, there is an object $o_1 \in B_1^{+\mathcal{I}}$ such that $(o_0, o_1) \in A_1^{\mathcal{I}}$. For $i \in \{2, \ldots, m\}$, by assertions $B_i^- \dot{\preceq} \forall A_1.B_i^-$ we have that $o_1 \in B_i^{-\mathcal{I}}$.

For the induction step, assume that the claim holds for all o_i with $i \in \{0, \ldots, u\}$. Let the binary representation of u be $b_m b_{m-1} \cdots b_1$. By induction hypothesis, if $b_i = 0$ then $o_u \in B_i^{-\mathcal{I}}$ and if $b_i = 1$ then $o_u \in B_i^{+\mathcal{I}}$. If $b_1 = b_2 = \cdots = b_m = 1$ the we are done. So let i_u be the least i such that $b_i = 0$. By assertion $B_{i_u}^- \dot{\preceq} \forall A_{i_u}.B_{i_u}^+ \sqcap \exists A_{i_u}$, there is an object $o_{u+1} \in B_{i_u}^{+\mathcal{I}}$ such that $(o_u, o_{u+1}) \in A_{i_u}^{\mathcal{I}}$. For $i \in \{1, \ldots, i_u - 1\}$, by assertions $B_i^+ \dot{\preceq} \forall A_{i_u}.B_i^-$ we have that $o_{u+1} \in B_i^{-\mathcal{I}}$. For $i \in \{i_u + 1, \ldots, m\}$, by assertions $B_i^- \dot{\preceq} \forall A_{i_u}.B_i^-$ and $B_i^+ \dot{\preceq} \forall A_{i_u}.B_i^+$ we have that if $o_u \in B_i^{-\mathcal{I}}$ then also $o_{u+1} \in B_i^{-\mathcal{I}}$, and if $o_u \in B_i^{+\mathcal{I}}$ then also $o_{u+1} \in B_i^{+\mathcal{I}}$. □

The objects mentioned in Lemma 4.2.17 are particular, since for each $i \in \{1, \ldots, m\}$ they are instances either of B_i^+ or of B_i^-. Observe, however, that each model $\mathcal{I}$ of $\mathcal{S}_m$ will contain also other objects that do not satisfy this condition. In particular, let o_u be an object in $\Delta^{\mathcal{I}}$. Then, for each $i \in \{1, \ldots, m\}$ such that $o_u \in B_i^{-\mathcal{I}}$, by assertion $B_i^- \dot{\preceq} \forall A_i.B_i^+ \sqcap \exists A_i$ there is an object $o_u^i \in B_i^{+\mathcal{I}}$ connected to o_u via role A_i. If i_0 is the least i, such that $o_u \in B_i^{-\mathcal{I}}$, the object $o_u^{i_0}$ is precisely o_{u+1}. For different values of i, however, the assertions neither force o_u^i to be an instance of $B_{i_0}^-$ nor to be an instance of $B_{i_0}^+$. Therefore we can construct a model of $\mathcal{S}_m$ in which o_{2^m-1} is the only common instance of all classes B_i^+. Moreover, all objects o_u can be pairwise distinct. We call a model satisfying these conditions *canonical* for $\mathcal{S}_m$. We state these observations as a lemma without proof for future reference.

Lemma 4.2.18 *Let $\mathcal{S}_m$ be the schema obtained as specified in Construction 4.2.16. Then $\mathcal{S}_m$ admits a particular model, in which the objects $o_0, o_1, \ldots, o_{2^m-1}$ are pairwise distinct, and in which o_{2^m-1} is the only object in $\Delta^{\mathcal{I}}$ that is a common instance of all classes B_i^+, for $i \in \{1, \ldots, m\}$.* □

We give now an intuitive description of the way we extend the schema $\mathcal{S}_m$ in order to obtain the reduction from validity of a quantified boolean formula $Q.f$ to class consistency in general primitive $\mathcal{L}_0$-schemata.

First of all we introduce additional classes and assertions that are used for "initializing the counter", i.e. forcing the existence of a common instance of all classes B_i^-, and for "checking termination", i.e. forcing a distinguished class to be populated if and only if the count proceeds till the end and there is a common instance of all classes B_i^+. We sketch the required additions referring to schema $\mathcal{S}_m$, although their final form in the schema that encodes the formula will be slightly different: The schema contains additionally an attribute A_0, the classes C_s, C_e, m^2 classes F_{ij}, for $i, j \in \{1, \ldots, m\}$, and the following assertions:

$$\begin{array}{rcll} C_s & \dot{\preceq} & \exists A_0 \sqcap \forall A_0.B_1^- \sqcap \cdots \sqcap \forall A_0.B_m^- & \\ B_i^+ & \preceq & \forall A_0.F_{i1}, & \text{for } i \in \{1, \ldots, m\} \\ B_1^+ & \preceq & \exists A_0 & \\ F_{ij} & \preceq & \forall A_0.F_{(i+1)j}, & \text{for } i \in \{1, \ldots, m-1\}, \\ & & & \quad j \in \{1, \ldots, m\} \\ F_{i(i+1)} & \dot{\preceq} & \exists A_0, & \text{for } i \in \{1, \ldots, m-1\} \\ F_{mi} & \preceq & \forall A_0.C_e, & \text{for } i \in \{1, \ldots, m\} \\ F_{m1} & \preceq & \exists A_0. & \end{array} \tag{4.18}$$

Assertions 4.18 are chosen in such a way that the set of chains starting at B_i^+ and ending in C_e is active and that no proper subset of this set of chains is active. This justifies the following lemma.

Lemma 4.2.19 *(1) Let $\mathcal{I}$ be a model of assertions 4.18 containing an object o with $o \in B_i^{+\mathcal{I}}$, for $i \in \{1, \ldots, m\}$. Then $C_e^{\mathcal{I}} \neq \emptyset$.*

(2) For each proper subset B of $\{1, \ldots, m\}$ there is a model I_B of assertions 4.18 containing an object o with $o \in B_i^{+\mathcal{I}}$, for $i \in B$, and $o \notin B_i^{+\mathcal{I}}$, for $i \notin B$, and $C_e^{\mathcal{I}} = \emptyset$.

Proof. We define the A_0-chains $F_i := B_i^+ F_{i1} \cdots F_{im} C_e$, for $i \in \{1, \ldots, m\}$. Assertions 4.18 are such that the set of A_0-chains $F := \{F_1, \ldots, F_m\}$ is active and every proper subset of F is not active. Claim 1 follows directly by applying Lemma 4.2.1, and claim 2 by applying Lemma 4.2.2. □

We then encode the clauses of f by means of chains obtained in a way similar to Construction 4.2.5, and we embed these chains in the schema that realizes the counter. The two chains K_i^+ and K_i^- that encode the clauses of f for literals i and $-i$ are inserted in the part of the schema that corresponds to the i-th bit of the counter. The idea is that these chains should block the count if there is a set $\mathbf{A}$ of Q-assignments canonical for Q and for which all assignments in $\mathbf{A}$ satisfy f. If such a set does not exist the count proceeds till the end and class C_e is forced to be populated. We ensure that in such a case an inconsistency occurs by adding an assertion which makes C_e inconsistent.

In order for the reduction to be effective, all sets of Q-assignments canonical for Q have to be tried in succession. If a letter i is universally quantified in Q, then each such set $\mathbf{A}$ contains pairs of assignments that assign the same values to $i{+}1, \ldots, m$, a different value to i, and are unrelated with respect to the values they assign to $1, \ldots, i{-}1$. Since both assignments that constitute such a pair have to satisfy f, if q is valid the count for both has to be blocked. Therefore, we insert the two chains K_i^+ and K_i^- corresponding to i in parallel in the schema, which ensures the generation of two different objects that separately activate the count. Conversely, if i is existentially quantified, then the sets of Q-assignments come in pairs containing the same assignments except for the value assigned to i by the assignments that correspond each other in the two sets. Therefore we have to try the two sets in succession and we insert the two chains that encode i in succession in the schema: K_i^- connected to B_i^-, and K_i^+ connected to B_i^+. This ensures that they are both traversed as the count proceeds and can separately contribute to block it.

Let $q := Q.f$ be a quantified boolean formula, where $Q := (Q_m m) \cdots (Q_1 1)$ is a prefix from 1 to m and $f := \{c_1, \ldots, c_n\}$ is a formula in DNF over $\{1, \ldots, m\}$. Let further k be the number of "$\forall$" in Q. In the following, when we refer to the h-th "$\forall$" quantifier we mean the h-th symbol "$\forall$" that appears in Q when scanning from left to right.

Construction 4.2.20 The schema $S_q := (\mathcal{C}_q, \mathcal{A}_q, \mathcal{T}_q)$ corresponding to a quantified boolean formula $q := Q.f$ is defined as follows:
The set $\mathcal{C}_q$ of classes of $\mathcal{S}_q$ contains the following elements:

- Two classes C_s, and C_e.
- m^2 classes F_{ij}, for $i, j \in \{1, \ldots, m\}$.
- $2{\cdot}m{\cdot}(n+1)$ classes C^+_{ij}, C^-_{ij}, for $i \in \{1, \ldots, m\}$, $j \in \{0, \ldots, n\}$. These classes correspond to the classes of Construction 4.2.5 that are used to encode the clauses of f.
- The classes that are used to implement the counter. For each "bit" i the number of these classes varies and depends also on the value of the i-th quantifier in Q. For this reason we introduce these classes below, when we need them in the assertions of the schema.

The set $\mathcal{A}_q$ of attributes of $\mathcal{S}_q$ contains the following elements:

- The attributes $A_1, \ldots, A_m$ used to implement the counter.
- The attribute A_c used for the encoding of the clauses.
- The attributes A^+, A^- used to connect the chains that encode the clauses to the schema that implements the counter.
- The attribute A_0 used to introduce the contradiction when the count terminates.

The set $\mathcal{T}_q$ of assertions of $\mathcal{S}_q$ contains the following elements:

- A set of assertions which are identical to the ones in Construction 4.2.5 and which encode the clauses of f. We repeat them here for completeness.
  - For each $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ the assertions
$$\begin{array}{lcl} C^+_{i(j-1)} & \dot{\preceq} & \forall A_c.C^+_{ij} \\ C^-_{i(j-1)} & \dot{\preceq} & \forall A_c.C^-_{ij}. \end{array} \tag{4.19}$$
  - For each $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$,
    - if $-i \in c_j$, the assertion
$$C^+_{i(j-1)} \;\dot{\preceq}\; \exists A_c. \tag{4.20}$$
    - if $i \in c_j$, the assertion
$$C^-_{i(j-1)} \;\dot{\preceq}\; \exists A_c. \tag{4.21}$$

For each $i \in \{1, \ldots, m\}$, the assertions that realize the i-th bit of the counter and the corresponding classes that have to be introduced depend on the value of Q_i, and if $Q_i = \forall$, also on the number of "$\forall$" that precede Q_i in Q. We remind that the total number of "$\forall$" in Q is k.

- If Q_i equals "$\exists$", then $\mathcal{C}_q$ contains $2{\cdot}k$ additional classes $B_i^{+1}, \ldots, B_i^{+k}$ and $B_i^{-1}, \ldots, B_i^{-k}$, and $\mathcal{T}_q$ contains the following assertions:
$$\begin{array}{lcl} B_i^{-j} & \dot{\preceq} & \forall A^-.B_i^{-(j+1)} \sqcap \forall A^+.B_i^{-(j+1)}, \\ & & \quad \text{for } j \in \{1, \ldots, k-1\} \\ B_i^{-k} & \dot{\preceq} & \forall A^-.C^-_{i0} \sqcap \forall A^+.C^-_{i0} \\ C^-_{in} & \dot{\preceq} & \forall A_1.B_i^{-1} \sqcap \forall A_2.B_i^{-1} \sqcap \cdots \sqcap \forall A_{i-1}.B_i^{-1} \\ C^-_{in} & \dot{\preceq} & \forall A_i.B_i^{+1} \sqcap \exists A_i \\ B_i^{+j} & \dot{\preceq} & \forall A^-.B_i^{+(j+1)} \sqcap \forall A^+.B_i^{+(j+1)}, \\ & & \quad \text{for } j \in \{1, \ldots, k-1\} \\ B_i^{+k} & \dot{\preceq} & \forall A^-.C^+_{i0} \sqcap \forall A^+.C^+_{i0} \\ C^+_{in} & \dot{\preceq} & \forall A_1.B_i^{+1} \sqcap \forall A_2.B_i^{+1} \sqcap \cdots \sqcap \forall A_{i-1}.B_i^{+1} \\ C^+_{in} & \dot{\preceq} & \forall A_{i+1}.B_i^{-1} \sqcap \forall A_{i+2}.B_i^{-1} \sqcap \cdots \sqcap \forall A_m.B_i^{-1} \\ B_i^{-1} & \dot{\preceq} & \neg B_i^{+1}. \end{array} \tag{4.22}$$

  These assertions have a form which is similar to that of assertions 4.17, considering that classes B_i^- and B_i^+ have been replaced with two chains of classes. These chains embed the chains that encode the clauses of f for literals i and $-i$ into the schema that implements the counter. In this case the chain that encodes the clauses for literal i is connected via some intermediate classes to the end of the chain that encodes the clauses for literal $-i$.

- If Q_i is the h-th "$\forall$", then $\mathcal{C}_q$ contains $h + 2{\cdot}k$ additional classes $B_i^1, \ldots, B_i^h$, $B_i^{+1}, \ldots, B_i^{+k}$, and $B_i^{-1}, \ldots, B_i^{-k}$, and $\mathcal{T}_q$ contains the following assertions:

$$\begin{array}{rcl}
B_i^j & \dot{\preceq} & \forall A^-.B_i^{j+1} \sqcap \forall A^+.B_i^{j+1}, \\
 & & \quad \text{for } j \in \{1, \ldots, h-1\} \\
B_i^h & \dot{\preceq} & \forall A^+.B_i^{+(h+1)} \sqcap \forall A^-.B_i^{-(h+1)} \\
B_i^{-j} & \dot{\preceq} & \forall A^-.B_i^{-(j+1)} \sqcap \forall A^+.B_i^{-(j+1)}, \\
 & & \quad \text{for } j \in \{1, \ldots, k-1\} \\
B_i^{-k} & \dot{\preceq} & \forall A^-.C_{i0}^- \sqcap \forall A^+.C_{i0}^- \\
C_{in}^- & \dot{\preceq} & \forall A_1.B_i^{-1} \sqcap \forall A_2.B_i^{-1} \sqcap \cdots \sqcap \forall A_{i-1}.B_i^{-1} \\
C_{in}^- & \dot{\preceq} & \forall A_i.B_i^1 \sqcap \forall A_{i+1}.B_i^1 \sqcap \cdots \sqcap \forall A_m.B_i^1 \\
C_{in}^- & \dot{\preceq} & \exists A_i \\
B_i^{+j} & \dot{\preceq} & \forall A^-.B_i^{+(j+1)} \sqcap \forall A^+.B_i^{+(j+1)}, \\
 & & \quad \text{for } j \in \{1, \ldots, k-1\} \\
B_i^{+k} & \dot{\preceq} & \forall A^-.C_{i0}^+ \sqcap \forall A^+.C_{i0}^+ \\
C_{in}^+ & \dot{\preceq} & \forall A_1.B_i^{+1} \sqcap \forall A_2.B_i^{+1} \sqcap \cdots \sqcap \forall A_{i-1}.B_i^{+1} \\
C_{in}^+ & \dot{\preceq} & \forall A_i.B_i^1 \sqcap \forall A_{i+1}.B_i^1 \sqcap \cdots \sqcap \forall A_m.B_i^1 \\
C_{in}^+ & \dot{\preceq} & \exists A_i \\
B_i^{-(h+1)} & \dot{\preceq} & \neg B_i^{+(h+1)}.
\end{array} \tag{4.23}$$

  In this case the chains that encode the clauses for literals i and $-i$ are inserted in parallel in the schema. The chain of classes $B_1 \cdots B_h$ and its increasing length with each "$\forall$"-quantifier in the prefix has the same effect of generation of objects achieved by the schema of Construction 4.2.3.

- A set of assertions which initialize the counter and force class C_e to be populated if the count terminates. These assertions are identical to assertions 4.18, except for the fact that in the present schema the classes B_i^- and B_i^+ have been replaced by classes that vary with the value of the i-th quantifier in the prefix.

  The assertions that do not depend on the quantifier prefix are:

$$\begin{array}{rcll}
F_{ij} & \dot{\preceq} & \forall A_0.F_{(i+1)j}, & \text{for } i \in \{1, \ldots, m-1\}, \\
 & & & \quad j \in \{1, \ldots, m\} \\
F_{i(i+1)} & \dot{\preceq} & \exists A_0, & \text{for } i \in \{1, \ldots, m-1\} \\
F_{mi} & \dot{\preceq} & \forall A_0.C_e, & \text{for } i \in \{1, \ldots, m\} \\
F_{m1} & \dot{\preceq} & \exists A_0. &
\end{array} \tag{4.24}$$

  For each $i \in \{1, \ldots, m\}$, the assertions that depend on the value of the i-th quantifier Q_i in the prefix are:

  – If Q_i equals "$\exists$"

$$\begin{array}{rcl}
C_s & \dot{\preceq} & \forall A_0.B_i^{-1} \\
C_{in}^+ & \dot{\preceq} & \forall A_0.F_{i1}.
\end{array} \tag{4.25}$$

  – If Q_i equals "$\forall$"

$$\begin{array}{rcl}
C_s & \dot{\preceq} & \forall A_0.B_i^1 \\
C_{in}^- & \dot{\preceq} & \forall A_0.F_{i1} \\
C_{in}^+ & \dot{\preceq} & \forall A_0.F_{i1}.
\end{array} \tag{4.26}$$

- We also need to add assertions that force the activation of all chains in the schema that involve the classes B_i^j, B_i^{+j} and B_i^{-j}, and the activation of the connection to the initial class C_s and the final chains constituted by the classes F_{ij}.

$$\begin{array}{rcll}
C_s & \dot{\preceq} & \exists A_0 & \\
C_{1n}^+ & \dot{\preceq} & \exists A_0 & \\
B_1^{+j} & \dot{\preceq} & \exists A^+ \sqcap \exists A^-, & \text{for } j \in \{1, \ldots, k\} \\
B_1^{-j} & \dot{\preceq} & \exists A^+ \sqcap \exists A^-, & \text{for } j \in \{1, \ldots, k\}.
\end{array} \tag{4.27}$$

If Q_1 equals "$\forall$", we add also the assertions:

$$\begin{array}{rcl} C_{1n}^- & \dot{\preceq} & \exists A_0 \\ B_1^1 & \dot{\preceq} & \exists A^+ \sqcap \exists A^-. \end{array} \tag{4.28}$$

- Finally, we add an assertion that forces an inconsistency in the schema if class C_e is forced to be populated.

$$C_e \ \dot{\preceq} \ \neg C_e. \tag{4.29}$$

■

Theorem 4.2.21 *Let q be a quantified boolean formula and $\mathcal{S}_q$ the primitive $\mathcal{L}_0$-schema obtained from q as specified in Construction 4.2.20. Then C_s is consistent in $\mathcal{S}_q$ if and only if q is valid.*

Proof (sketch). We first discuss the case where Q contains a single quantifier. If $q := \exists 1.f$, then $\mathcal{S}_q$ consists of a single chain from class C_s to class C_e, which contains as subchains the encodings K_1^+ and K_1^- of literals 1 and -1 in f. If q is valid there is a Q-assignment α that satisfies f, and since Q contains just one quantifier, either K_1^- or K_1^+ correspond to α. By Lemma 4.2.7 either K_1^- or K_1^+ is not active in $\mathcal{S}_q$, and so the whole chain is not active in $\mathcal{S}_q$. By Lemma 4.2.2 we can populate C_s in a model $\mathcal{I}$ of $\mathcal{S}_q$ without being forced to populate C_e. On the other hand, if q is not valid then both chains K_1^+ and K_1^- are active. By Lemma 4.2.1, if we try to populate class C_s we are forced to introduce an instance of class C_e, which generates a contradiction. Therefore C_s is not consistent in $\mathcal{S}_q$.

Similarly, if $q := \forall 1.f$, then assertions 4.23 give rise to two separate chains K_1^- and K_1^+ which both connect C_s to C_f. Note that K_1^- and K_1^+ cannot activate each other since K_1^- is an $(A_0A^-\cdots)$-chain, while K_1^+ is an $(A_0A^+\cdots)$-chain. If q is valid both possible Q-assignments satisfy f. Therefore neither K_1^- nor K_1^+ is active in $\mathcal{S}_q$ and C_s is consistent in $\mathcal{S}_q$. If q is not valid then at least one of the Q-assignments does not satisfy f and the corresponding chain is active. This forces the introduction of an instance of C_e as soon as we try to populate C_s.

For an arbitrary prefix $Q := (Q_1 1)\cdots(Q_m m)$, we attempt the construction of a model $\mathcal{I}$ of $\mathcal{S}_q$ in which $C_s^{\mathcal{I}} \neq \emptyset$ by stepping through the counter. If we put an object o in $C_s^{\mathcal{I}}$, by assertions 4.25.1, 4.26.1, and 4.27.1 we have to introduce an object o_0 that is an instance of all initial classes that implement the counter. By Lemma 4.2.17 this object forces in every model the introduction of a sequence of objects that correspond to the steps of the count. In particular, if we are forced to populate with a common instance o all final classes in the chains of the counter (for bit i, these classes are C_{in}^+ if Q_i equals "$\exists$", and either C_{in}^+ or C_{in}^-, if Q_i equals "$\forall$"), by part 1 of Lemma 4.2.19 we are also forced to populate C_e, and therefore to generate a contradiction. On the other hand, if we can block the count and avoid to generate such a common instance o, by part 2 of Lemma 4.2.19 we can also avoid to generate an instance of C_e. By Lemma 4.2.18 it is sufficient to consider a model of $\mathcal{S}_q$ that corresponds to the canonical model of $\mathcal{S}_m$, in which at each step of the count new objects are generated.

In the rest of the proof we motivate informally why the count can be blocked if and only if q is valid. One difference between $\mathcal{S}_q$ and the schema of Construction 4.2.16 that implements the simple counter is that in $\mathcal{S}_q$ the generation of new objects may in fact be blocked due to the set of the intermediate chains that encode the clauses. Assertions 4.22 and 4.23 are such that for each canonical set of Q-assignments $\mathbf{A}$ the following holds: Let H_α be the set of A_c-chains that correspond to a Q-assignment α (in the sense of Definition 4.2.6). Then for each α in $\mathbf{A}$ there is an object o_α that is an instance of all initial classes of the chains in H_α. If there is an α that does not satisfy f, then the set H_α is active, and the presence of o_α forces the existence of an object that represents the next step in the count. If all $\alpha \in \mathbf{A}$ satisfy f, then no set of chains H_α is active and we can stop the generation of new objects and therefore the count. Since all sets of Q-assignments canonical for Q are considered in succession, if there is one such set all of whose members satisfy f, then at some point the count is terminated and we are not forced to populate class C_e.

It remains to argue that in fact each set of Q-assignments canonical for Q is tried in succession, and that for each such set $\mathbf{A}$ all Q-assignments in $\mathbf{A}$ are considered in parallel. This can be shown by induction on the number m of quantifiers in the prefix. We have already discussed the case when we have just one quantifier. For the induction step, assume that for a prefix Q of length m we traverse all possible Q assignments in succession. If we now consider the prefix $(\exists m+1)Q$, we have to add to the counter the subschema defined by assertions 4.22, in which the chains K_{m+1}^- and K_{m+1}^+ are traversed in succession. Since this schema represents the most significant bit of the counter, while K_{m+1}^- is traversed 2^m times, by induction hypotheses all Q-assignments canonical for Q are considered in succession. The same holds while K_{m+1}^+ is traversed, which

shows the claim for this case. If we consider the prefix $(\forall m+1)Q$, we have to add to the counter the subschema 4.23, in which the chains K^-_{m+1} and K^+_{m+1} are traversed in parallel. In fact, assertion 4.23.2 guarantees that two distinct objects are generated, for which one causes the traversal of K^-_{m+1} and the other of K^+_{m+1}. Therefore, we obtain that the assignments α with $\alpha(m) = \mathbf{t}$ are considered in parallel with the assignments α with $\alpha(m) = \mathbf{f}$. We also remark that in order to guarantee that while K^-_{m+1} (K^+_{m+1}) is traversed all canonical sets of Q-assignments are considered, we had to introduce classes B^1_i, B^2_i, ..., B^h_i and the corresponding assertions that realize the chains of varying length for each "$\forall$"-quantifier. □

As an immediate consequence of Theorem 4.2.21 we get the following lower bound for class consistency in general primitive $\mathcal{L}_0$-schemata.

Corollary 4.2.22 *Class consistency in general primitive $\mathcal{L}_0$-schemata is **PSPACE**-hard.* □

In [37] an algorithm is given that decides class consistency in general primitive $\mathcal{L}_0$-schemata in polynomial space. This shows that the problem is **PSPACE**-complete.

Corollary 4.2.23 *Class consistency in general primitive $\mathcal{L}_0$-schemata is **PSPACE**-complete.* □

4.3 Eliminating Qualified Existential Quantification

In this section we develop a technique by which a schema, possibly containing the constructor for existential quantification, can be transformed into a schema where this constructor is not present any more. The transformation, which we call *$\mathcal{E}$-elimination*, is polynomial and it does not influence the consistency of the classes in the schema. It can be applied to primitive schemata, both cyclic and acyclic, expressed in $\mathcal{LC}$ or one of its sub-languages. Since the other constructors, and in particular disjunction, and the acyclicity of the schema are preserved, our technique provides immediately several results about reasoning on schemata expressed in the sub-languages of $\mathcal{LC}$ and about its computational complexity. On one hand, reasoning procedures developed for schemata expressed in $\mathcal{L}_0$ and $\mathcal{LU}$ can be used to reason on schemata expressed in the more expressive languages $\mathcal{LE}$ and $\mathcal{LC}$, respectively. On the other hand, we can deduce several new interesting lower bounds for the complexity of class consistency, the most relevant being **EXPTIME**-hardness of class consistency in general primitive $\mathcal{LU}$-schemata. We remind that primitive schemata represent the simplest form of schema since they allow one to state only sufficient conditions for membership in a class. However, the use of disjunction together with cyclic assertions in the schema is already sufficient to make reasoning on primitive schemata as hard as in the most expressive (but still decidable) schema definition languages we consider in this thesis.

$\mathcal{E}$-elimination is based on an idea used in [37] to show that verifying class consistency in $\mathcal{L}_0$-schemata is **coNP**-hard. The result is established by exploiting a reduction from subsumption between class expressions in $\mathcal{LE}$, which was shown **coNP**-complete in [62], to class consistency in $\mathcal{L}_0$-schemata. The reduction is based essentially on the elimination of existential quantification from an $\mathcal{LE}$ class expression containing a single attribute by introducing suitable assertions in a schema. We show how to generalize this transformation to an arbitrary number of attributes and to arbitrary primitive $\mathcal{LC}$-schemata.

In order to define $\mathcal{E}$-elimination we need to perform a preliminary transformation on the schema. An $\mathcal{LC}$-class expression is called *normalized* if it is of the form:

$$C \mid \neg C \mid C_1 \sqcup C_2 \mid \forall A.C \mid \exists A \mid \exists A.C.$$

An assertion is said to be *normalized*, if the class expression on the right-hand side is normalized. A primitive $\mathcal{LC}$-schema is *normalized* if it is constituted solely by normalized class specifications.

Lemma 4.3.1 *Every primitive $\mathcal{LC}$-schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ can be transformed in linear time into an equivalent normalized primitive $\mathcal{LC}$-schema $\mathcal{S}' := (\mathcal{C}', \mathcal{A}, \mathcal{T}')$, where $\mathcal{C} \subseteq \mathcal{C}'$.*

Proof. The normalized schema $\mathcal{S}'$ equivalent to $\mathcal{S}$ can be obtained by setting $\mathcal{C}' := \mathcal{C}$ and $\mathcal{T}' := \mathcal{T}$, and repeating the following steps until all assertions in $\mathcal{T}'$ are normalized: For each assertion $C \doteq E$ in $\mathcal{T}'$, if E is not normalized then remove $C \doteq E$ from $\mathcal{T}'$ and do the following:

- If $E = E_1 \sqcap E_2$, then add $C \dot{\preceq} E_1$ and $C \dot{\preceq} E_2$ to $\mathcal{T}'$.
- If $E = E_1 \sqcup E_2$, then add two new class names C_1 and C_2 to $\mathcal{C}'$ and add $C \dot{\preceq} C_1 \sqcup C_2$, $C_1 \dot{\preceq} E_1$, and $C_2 \dot{\preceq} E_2$ to $\mathcal{T}'$.
- If $E = \forall A.E'$, then add a new class name C_1 to $\mathcal{C}'$ and add $C \dot{\preceq} \forall A.C_1$ and $C_1 \dot{\preceq} E'$ to $\mathcal{T}'$.
- If $E = \exists A.E'$, then add a new class name C_1 to $\mathcal{C}'$ and add $C \dot{\preceq} \exists A.C_1$ and $C_1 \dot{\preceq} E'$ to $\mathcal{T}'$.

The above construction introduces at most one new class name for each subexpression of a class expression in $\mathcal{T}$. Since the number of such subexpressions is linear in $|\mathcal{T}|$, at most a linear number of new classes is introduced, and $|\mathcal{S}'|$ is linear in $|\mathcal{S}|$. It is also easy to see that the construction can be performed in linear time in $|\mathcal{S}|$. By construction $\mathcal{S}'$ is normalized and $\mathcal{C} \subseteq \mathcal{C}'$.

It remains to show that $\mathcal{S}$ and $\mathcal{S}'$ are equivalent. We proceed by induction on the number k of applied steps of the above procedure, showing that after each step the resulting schema is equivalent to $\mathcal{S}$.
"$k = 0$": Then $\mathcal{C}' = \mathcal{C}$, $\mathcal{T}' = \mathcal{T}$, and $\mathcal{S}$ and $\mathcal{S}'$ are identical.
"$k \to k+1$": Let $\mathcal{S}_k := (\mathcal{C}_k, \mathcal{A}, \mathcal{T}_k)$ be obtained from $\mathcal{S}$ by interrupting the above procedure after k new class names have been introduced, and assume $\mathcal{C}_k := \mathcal{C} \cup \{C_1, \ldots, C_h\}$. Let the next step of the procedure be applied to $C \dot{\preceq} \forall A.E$, where E is not normalized, and let $\mathcal{S}_{k+1} := (\mathcal{C}_{k+1}, \mathcal{A}, \mathcal{T}_{k+1})$ be the resulting schema, where $\mathcal{C}_{k+1} := \mathcal{C}_k \cup \{C_{h+1}\}$ and $\mathcal{T}_{k+1} := \mathcal{T}_k \cup \{C_{h+1} \dot{\preceq} E,\ C \dot{\preceq} \forall A.C_{h+1}\} \setminus \{C \dot{\preceq} \forall A.E\}$. By induction hypothesis, $\mathcal{S}_k$ is equivalent to $\mathcal{S}$. Let $\mathcal{I}$ be a model of $\mathcal{S}$, and $\mathcal{I}_k$ a model of $\mathcal{S}_k$ that extends $\mathcal{I}$ by interpreting $C_1, \ldots, C_h$ in an appropriate way. We extend $\mathcal{I}_k$ to a model $\mathcal{I}_{k+1}$ of $\mathcal{S}_{k+1}$, by setting $C_{h+1}^{\mathcal{I}_{k+1}} := E^{I_k}$. $\mathcal{I}_{k+1}$ extends $\mathcal{I}_k$ and hence also $\mathcal{I}$, and since $\mathcal{I}_k$ satisfies $C \dot{\preceq} \forall A.E$, $\mathcal{I}_{k+1}$ satisfies by construction also both newly introduced assertions and is therefore a model of $\mathcal{S}_{k+1}$. Let now $\mathcal{I}_{k+1}$ be a model of $\mathcal{S}_{k+1}$. Since it satisfies $C \dot{\preceq} \forall A.C_{h+1}$, for every instance o of C, if o is connected via attribute A to an object o', then o' is an instance of C_{h+1}. Since $\mathcal{I}_{k+1}$ satisfies also $C_{h+1} \dot{\preceq} E$, o' is also an instance of class expression E. Therefore $\mathcal{I}_{k+1}$ satisfies $C \dot{\preceq} \forall A.E$ and it is a model of $\mathcal{S}_k$. By induction hypothesis, every model of $\mathcal{S}_k$, and hence also $\mathcal{I}_{k+1}$ is also a model of $\mathcal{S}$. In case the next step of the procedure is applied to $\exists A.E$ or $C \dot{\preceq} E_1 \sqcup E_2$ we can proceed in a similar way. □

Lemmata 4.3.1 and 2.3.7 allow us to restrict our attention to normalized schemata, and in the rest of the section we assume that the schemata we deal with are all normalized primitive $\mathcal{LC}$-schemata.

Construction 4.3.2 Let $\mathcal{S}_\mathcal{E} := (\mathcal{C}_\mathcal{E}, \mathcal{A}_\mathcal{E}, \mathcal{T}_\mathcal{E})$ be a normalized $\mathcal{LC}$-schema. The schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ obtained from $\mathcal{S}_\mathcal{E}$ by *$\mathcal{E}$-elimination* is defined as follows:

- $\mathcal{C} := \mathcal{C}_\mathcal{E}$.
- For an attribute $A \in \mathcal{A}_\mathcal{E}$, let $\mathcal{A}_A := \{A_C \mid \exists A.C \text{ appears in } \mathcal{T}_\mathcal{E}\}$. Then $\mathcal{A} := \mathcal{A}_\mathcal{E} \cup \bigcup_{A \in \mathcal{A}_\mathcal{E}} \mathcal{A}_A$.
- $\mathcal{T}$ is obtained from $\mathcal{T}_\mathcal{E}$ by substituting each assertion

$$\begin{array}{llll} C \dot{\preceq} \exists A.C' & \text{with} & C \dot{\preceq} \exists A_{C'} \sqcap \forall A_{C'}.C', & \text{and} \\ C \dot{\preceq} \forall A.C' & \text{with} & C \dot{\preceq} \forall A.C' \sqcap \forall A_{C_1}.C' \sqcap \cdots \sqcap \forall A_{C_n}.C', & \\ & & \text{where } \mathcal{A}_A = \{A_{C_1}, \ldots, A_{C_n}\}, & \end{array}$$

  and leaving the other assertions unchanged.

We observe that the number of new attributes introduced in $\mathcal{S}$ is linear in the number of assertions in $\mathcal{S}_\mathcal{E}$, and that $|\mathcal{S}|$ is at most quadratic in $|\mathcal{S}_\mathcal{E}|$. Moreover, the constructor for disjunction is present in $\mathcal{S}$ if and only if it is present in $\mathcal{S}_\mathcal{E}$. Although we have replaced each qualified existential quantification with an unqualified one on a different attribute, the interaction between universal and existential quantification is still captured correctly. In fact, the following theorem holds.

Theorem 4.3.3 *Let $\mathcal{S}_\mathcal{E}$ be an $\mathcal{LC}$-schema, $C \in \mathcal{C}_\mathcal{E}$, and $\mathcal{S}$ be the schema obtained from $\mathcal{S}_\mathcal{E}$ by $\mathcal{E}$-elimination. Then C is consistent in $\mathcal{S}$ if and only if it is consistent in $\mathcal{S}_\mathcal{E}$.*

Proof. "⇐" Let $\mathcal{I}_\mathcal{E}$ be a model of $\mathcal{S}_\mathcal{E}$ with $C^{\mathcal{I}_\mathcal{E}} \neq \emptyset$. We define an interpretation $\mathcal{I}$ of $\mathcal{S}$ as follows:

- $\Delta^\mathcal{I} := \Delta^{\mathcal{I}_\mathcal{E}}$.

- For every class $C \in \mathcal{C}$, $C^{\mathcal{I}} := C^{\mathcal{I}_\mathcal{E}}$.
- For every attribute $A \in \mathcal{A}_\mathcal{E}$, $A^{\mathcal{I}} := A^{\mathcal{I}_\mathcal{E}}$, and for every attribute $A_C \in \mathcal{A}_A$, $A_C^{\mathcal{I}} := A^{\mathcal{I}_\mathcal{E}} \cap (\Delta^{\mathcal{I}_\mathcal{E}} \times C^{\mathcal{I}_\mathcal{E}})$.

We show that $\mathcal{I}$ is a model of $\mathcal{S}$, by showing that it satisfies every assertion in $\mathcal{T}$. Let $C \doteq E$ be such an assertion:

- If $E = C'$, $E = \neg C'$, or $E = C_1 \sqcup C_2$, then $(C \doteq E) \in \mathcal{T}_\mathcal{E}$. Therefore $C^{\mathcal{I}_\mathcal{E}} \subseteq E^{\mathcal{I}_\mathcal{E}}$ and by definition of $\mathcal{I}$ the assertion is satisfied.
- If $E = \exists A$, for some $A \in \mathcal{A}_\mathcal{E}$, then $(C \doteq \exists A) \in \mathcal{T}_\mathcal{E}$. Since by definition of $\mathcal{I}$, $C^{\mathcal{I}} = C^{\mathcal{I}_\mathcal{E}}$ and $A^{\mathcal{I}} = A^{\mathcal{I}_\mathcal{E}}$, $\mathcal{I}$ satisfies $C \doteq \exists A$.
- If $E = \exists A_{C'} \sqcap \forall A_{C'}.C'$, then $(C \doteq \exists A.C') \in \mathcal{T}_\mathcal{E}$. The assertion $C \doteq \forall A_{C'}.C'$ is satisfied since by definition $A_{C'}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}_\mathcal{E}} \times C'^{\mathcal{I}_\mathcal{E}}$. Let $o \in C^{\mathcal{I}}$. Since $\mathcal{I}_\mathcal{E}$ satisfies $C \doteq \exists A.C'$ and $o \in C^{\mathcal{I}_\mathcal{E}}$ there is an object $o' \in C'^{\mathcal{I}_\mathcal{E}}$ such that $(o, o') \in A^{\mathcal{I}_\mathcal{E}}$. By definition of $A_{C'}^{\mathcal{I}}$, we have that $(o, o') \in A_{C'}^{\mathcal{I}}$, and $\mathcal{I}$ satisfies $C \doteq E$.
- If $E = \forall A.C' \sqcap \forall A_{C_1}.C' \sqcap \cdots \sqcap \forall A_{C_n}.C'$, then $\mathcal{A}_A = \{A_{C_1}, \ldots, A_{C_n}\}$ and $(C \doteq \forall A.C') \in \mathcal{T}_\mathcal{E}$. The assertion $C \doteq \forall A.C'$ is satisfied since by definition $C^{\mathcal{I}} = C^{\mathcal{I}_\mathcal{E}}$, $C'^{\mathcal{I}} = C'^{\mathcal{I}_\mathcal{E}}$, and $A^{\mathcal{I}} = A^{\mathcal{I}_\mathcal{E}}$. We show that $C \doteq \forall A_{C_i}.C'$ is satisfied in $\mathcal{I}$, for $i \in \{1, \ldots, n\}$. Assume by contradiction that there are two objects $o \in C^{\mathcal{I}}$ and $o' \notin C'^{\mathcal{I}}$ such that $(o, o') \in A_{C_i}^{\mathcal{I}}$. Since $A_{C_i} \in \mathcal{A}_A$, by definition $A_{C_i}^{\mathcal{I}} \subseteq A^{\mathcal{I}_\mathcal{E}}$, and we have also that $(o, o') \in A^{\mathcal{I}_\mathcal{E}}$. Therefore $\mathcal{I}_\mathcal{E}$ does not satisfy $C \doteq \forall A.C'$.

"$\Rightarrow$" Let $\mathcal{I}$ be a model of $\mathcal{S}$ with $C^{\mathcal{I}} \neq \emptyset$. We define an interpretation $\mathcal{I}_\mathcal{E}$ of $\mathcal{S}_\mathcal{E}$ as follows:

- $\Delta^{\mathcal{I}_\mathcal{E}} := \Delta^{\mathcal{I}}$.
- For every class $C \in \mathcal{C}_\mathcal{E}$, $C^{\mathcal{I}_\mathcal{E}} := C^{\mathcal{I}}$.
- For every attribute $A \in \mathcal{A}_\mathcal{E}$ with $\mathcal{A}_A = \{A_{C_1}, \ldots, A_{C_n}\}$, $A^{\mathcal{I}_\mathcal{E}} := A^{\mathcal{I}} \cup A_{C_1}^{\mathcal{I}} \cup \cdots \cup A_{C_n}^{\mathcal{I}}$.

We show that $\mathcal{I}_\mathcal{E}$ is a model of $\mathcal{S}_\mathcal{E}$, by showing that it satisfies every assertion in $\mathcal{T}_\mathcal{E}$. Let $C \doteq E$ be such an assertion, where E is a normalized class expression:

- If $E = C'$, $E = \neg C'$, or $E = C_1 \sqcup C_2$, then $(C \doteq E) \in \mathcal{T}$. Therefore $C^{\mathcal{I}} \subseteq E^{\mathcal{I}}$ and by definition of $\mathcal{I}_\mathcal{E}$ the assertion is satisfied.
- If $E = \exists A$, then $(C \doteq \exists A) \in \mathcal{T}_\mathcal{E}$. Since by definition of $\mathcal{I}_\mathcal{E}$, $C^{\mathcal{I}_\mathcal{E}} = C^{\mathcal{I}}$ and $A^{\mathcal{I}_\mathcal{E}} \subseteq A^{\mathcal{I}}$, $\mathcal{I}_\mathcal{E}$ satisfies $C \doteq \exists A$.
- If $E = \exists A.C'$, then $(C \doteq \exists A_{C'} \sqcap \forall A_{C'}.C') \in \mathcal{T}$. Let $o \in C^{\mathcal{I}_\mathcal{E}}$. Since $\mathcal{I}$ satisfies $C \doteq \exists A_{C'}$ and $o \in C^{\mathcal{I}}$ there is an object $o' \in \Delta^{\mathcal{I}}$ such that $(o, o') \in A_{C'}^{\mathcal{I}}$, and since $\mathcal{I}$ satisfies $C \doteq \forall A_{C'}.C'$, $o' \in C'^{\mathcal{I}}$. Since $A_{C'} \in \mathcal{A}_A$, by definition of $A^{\mathcal{I}_\mathcal{E}}$, we have that $(o, o') \in A^{\mathcal{I}_\mathcal{E}}$, and since $o' \in C'^{\mathcal{I}}$, $\mathcal{I}_\mathcal{E}$ satisfies $C \doteq \exists A.C'$.
- If $E = \forall A.C'$, then $(C \doteq \forall A.C' \sqcap \forall A_{C_1}.C' \sqcap \cdots \sqcap \forall A_{C_n}.C') \in \mathcal{T}_\mathcal{E}$, where $\mathcal{A}_A = \{A_{C_1}, \ldots, A_{C_n}\}$. Assume by contradiction that there are two objects $o \in C^{\mathcal{I}_\mathcal{E}}$ and $o' \notin C'^{\mathcal{I}_\mathcal{E}}$ such that $(o, o') \in A^{\mathcal{I}_\mathcal{E}}$. By definition of $A^{\mathcal{I}_\mathcal{E}}$, $(o, o') \in A^{\mathcal{I}}$ or $(o, o') \in A_{C_i}^{\mathcal{I}}$ for some $i \in \{1, \ldots, n\}$. Since $o \in C^{\mathcal{I}}$ and $o' \notin C'^{\mathcal{I}}$, in the first case $\mathcal{I}$ does not satisfy $C \doteq \forall A.C'$ and in the second case $\mathcal{I}$ does not satisfy $C \doteq \forall A_{C_i}.C'$.

□

An immediate consequence of Theorem 4.3.3 is a complete characterization of the complexity of verifying class consistency in primitive $\mathcal{LE}$-schemata, both acyclic and general.

Corollary 4.3.4 *(1) Class consistency in acyclic primitive $\mathcal{LE}$-schemata is **coNP**-complete. (2) Class consistency in general primitive $\mathcal{LE}$-schemata is **PSPACE**-complete.*

Proof. By Lemmata 4.3.1 and 2.3.7 we can assume without loss of generality to deal only with normalized $\mathcal{LE}$-schemata. For a primitive $\mathcal{LE}$-schema $\mathcal{S}_\mathcal{E}$, the schema $\mathcal{S}$ obtained from $\mathcal{S}_\mathcal{E}$ by applying Construction 4.3.2 does not contain disjunction and is therefore a primitive $\mathcal{L}_0$-schema. Moreover, $|\mathcal{S}|$ is polynomial in $|\mathcal{S}_\mathcal{E}|$, and $\mathcal{S}$ is acyclic if and only if $\mathcal{S}_\mathcal{E}$ is so. Therefore, by Theorem 4.3.3 we can polynomially reduce class consistency in $\mathcal{LE}$ to class consistency in $\mathcal{L}_0$. The claims follow from Corollaries 4.2.11 and 4.2.23, which state that class consistency in primitive $\mathcal{L}_0$-schemata is **coNP**-complete for acyclic schemata and **PSPACE**-complete for general schemata. □

We get also a further confirmation of the fact that the complexity source introduced by (acyclic) assertions in a schema is of the same nature as the one introduced by existential quantification, and results precisely from the interplay between (unqualified) existential quantification and universal quantification.

Similarly, $\mathcal{E}$-elimination allows us to show lower bounds on reasoning on $\mathcal{LU}$-schemata, by exploiting the lower bounds for $\mathcal{LC}$. In particular, since verifying the consistency of $\mathcal{LC}$-class expressions is **PSPACE**-hard (see [143]) the same hardness result clearly holds for acyclic primitive $\mathcal{LC}$-schemata. Using $\mathcal{E}$-elimination and applying Theorem 4.3.3 we obtain therefore an alternative proof of Theorem 4.2.15. More important, from **EXPTIME**-completeness of class consistency in general primitive $\mathcal{LC}$-schemata (see [39]), we obtain the same completeness result for general primitive $\mathcal{LU}$-schemata, of which the hardness part was open.

Corollary 4.3.5 *Class consistency in general primitive $\mathcal{LU}$-schemata is **EXPTIME**-hard.*

Proof. The proof is similar to the proof of Corollary 4.3.4, using **EXPTIME**-hardness of class consistency in general primitive $\mathcal{LC}$-schemata. □

We would like to point out that although $\mathcal{E}$-elimination has proved useful for the schema definition languages below $\mathcal{LC}$, it cannot easily be extended to deal with the other constructors introduced in Chapter 2. In the schema obtained by $\mathcal{E}$-elimination each attribute A of the original schema is replaced by a set $\mathcal{A}_A$ of new attributes. Therefore, each number restriction on A appearing in the schema has to be replaced by a number restriction on the union of all attributes in $\mathcal{A}_A$, which leads to introduce in the resulting schema the constructor for union of attributes. Similarly, if inverse attributes are present, it is not clear how an assertion of the form $C \doteq \exists A^{-}.C'$, involving a qualified existential quantification over an inverse attribute should be treated.

4.4 Undecidable $\mathcal{L}$-Languages

In this section we show that the expressivity of some of the $\mathcal{L}$-languages introduced in Chapter 2 makes reasoning about class expressions and/or schemata undecidable. The undecidability is a consequence of the presence and interaction of several problematic constructors due to the possibility to combine these without any restrictions. We will see in Section 5.2 that the limitations introduced in $\mathcal{LT}^{-}$ on the use of these constructors are sufficient to make reasoning decidable on all types of schemata.

The following theorem proved in [142] shows that the addition to $\mathcal{L}_0$ of role value maps alone already makes verifying subsumption between class expressions undecidable, even if the schema is empty (and therefore reasoning is performed with respect to any interpretation).

Theorem 4.4.1 (Schmidt-Schauß [142]) *Subsumption between class expressions in $\mathcal{LV}$ is undecidable.* □

As shown in [87, 88], allowing the use of unrestricted intersection of attributes in $\mathcal{LCFL}$ is sufficient to make the problem of verifying the consistency of a class expression highly undecidable[3]. The proof in [87] exploits a reduction to variants of the classic unbounded domino problems, which we very briefly introduce here.

A *domino* D is a 1×1 square tile fixed in orientation, each of whose edges is associated with some color. A *domino problem* is a decision problem that asks whether or not it is possible to tile some portion of the integer grid $Z \times Z$, given a finite set of domino types. The general rule for tiling is that each point of the grid is associated with one tile type and that adjacent tiles have the same color on the common edge. Each domino problem may specify additional constraints that the tiling has to satisfy. As shown in [26, 134] the problem that asks whether a given set of tile types can tile the positive quadrant of the integer grid is already undecidable[4].

Domino problems provide a particularly elegant method for proving lower bounds for program and modal logics, and thus also for $\mathcal{L}$-languages. Harel in [87] shows that the key point in such proofs is the possibility in the logic to define the two-dimensional integer grid on which the tiles have to be placed. Once the grid is defined, to prove undecidability it is sufficient to be able to "access" all points on the grid and state

[3] By highly undecidable we mean that this problem is Σ_1^1-complete.
[4] More precisely, the unrestricted tiling problem is Π_1^0-complete.

properties for them. An analysis of the (very simple and elegant) proof in [87] shows that the undecidability result in the presence of intersection holds already for a simpler variant of the logic. This allows us to state the following theorem.

Theorem 4.4.2 *If we add to a primitive $\mathcal{LUF}$-schema a single assertion of the form $C \dot{\preceq} \exists((A \circ A') \cap (A' \circ A))$, then class consistency becomes undecidable.*

Proof (sketch). The grid can be defined through the following Schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$:

$\mathcal{C} :=$ {Point}
$\mathcal{A} :=$ {right,up}

$\mathcal{T} :=$ {Point $\dot{\preceq}$ $\exists^{=1}$right $\sqcap$ $\forall$right.Point $\sqcap$ $\exists^{=1}$up $\sqcap$ $\forall$up.Point
Point $\dot{\preceq}$ $\exists((\text{right} \circ \text{up}) \cap (\text{up} \circ \text{right}))$}

The assertion involving the intersection of the two chains of (functional) attributes, forces any model of this schema either to be a grid or to be such that it can be unwound to one. Given a set of tile types, the assertions that express the tiling constraints can easily be added (together with the necessary classes) to this schema as shown in [87]. They require only to assert that each point in the grid should get one of the tiles (which can be done using disjunction), and ensuring that adjacent tiles have the same color on the common edge (which can be done using disjunction and universal quantification on the two attributes right and up).

Then the class Point is consistent in the resulting schema if and only if the positive quadrant can be tiled with the given set of tile types. □

Undecidability arises also if we allow the unrestricted use of number restrictions, and in particular of functional restrictions on complex links.

Theorem 4.4.3 *If we add to a primitive $\mathcal{LUF}$-schema a single assertion of the form $C \dot{\preceq} \exists^{\leq 1}((A \circ A') \cup (A' \circ A))$, then class consistency becomes undecidable.*

Proof (sketch). The definition of the grid is similar to the one in the proof of Theorem 4.4.2, except that we replace the assertion

$$\text{Point} \dot{\preceq} \exists((\text{right} \circ \text{up}) \cap (\text{up} \circ \text{right}))$$

with

$$\text{Point} \dot{\preceq} \exists^{\leq 1}((\text{right} \circ \text{up}) \cup (\text{up} \circ \text{right})).$$

□

Theorems 4.4.1, 4.4.2, and 4.4.3 show that in order to obtain from $\mathcal{LT}$ a language in which reasoning on a schema is decidable, we must at least restrict the use of all problematic constructors. This is achieved in $\mathcal{LT}^-$, where only a limited interaction of number restrictions, difference (and therefore intersection) of links, and role value maps is allowed.

4.5 Discussion

In this chapter we have dealt with the intrinsic complexity of reasoning on primitive $\mathcal{L}$-schemata. We now discuss briefly previous work on the topic, carried out mostly in the field of Description Logics, and the impact of the results we established.

We mentioned already that while the complexity of reasoning on class expressions has been thoroughly investigated, deduction on schemata, was not so well understood. This is especially true of primitive schemata, i.e. schemata containing only primitive class specifications and no definitions.

For non primitive schemata, reasoning is already complex, even if the language used for constructing class expressions is very limited. The complexity of class subsumption in such types of schemata for a very simple language, containing only the constructors for class conjunction and universal quantification over attributes, is investigated by Baader [9] and Nebel [122]. They reduce the problem of subsumption between two classes with respect to a (possibly cyclic) schema containing only class definitions to decision problems for various types of finite sequential automata. In particular, Baader shows that when least or greatest

Schema / Language	empty	primitive	
		acyclic	general
$\mathcal{L}^-$ (subsum.)	**PTIME** [32]	**PTIME** [40]	**PTIME** [40]
$\mathcal{L}_0$	**PTIME** [143]	**coNP** [37], Cor. 4.2.10	**PSPACE** [37], Cor. 4.2.22
$\mathcal{LE}$	**coNP** [62]	**coNP** Cor. 4.3.4 (1)	**PSPACE** Cor. 4.3.4 (2)
$\mathcal{LU}$	**NP** [63]	**PSPACE**-hard Cor. 4.2.15	**EXPTIME** Cor. 4.3.5
$\mathcal{LC}$	**PSPACE** [143]	**PSPACE**-hard [143]	**EXPTIME** [39]
$\mathcal{LCL}$	**EXPTIME** [127, 140]	**EXPTIME** [107, 140]	**EXPTIME** [107, 140]

Table 4.1: Computational complexity of class consistency in primitive schemata

fixpoint semantics is adopted, class subsumption can be reduced to a problem of language inclusion for two finite state automata on finite words. This problem is **coNP**-complete if the automaton, and therefore the schema is acyclic, while it becomes **PSPACE**-complete for general schemata. If descriptive semantics is adopted, however, the reduction is more involved and leads to a problem of language inclusion for Büchi sequential automata (see Appendix A). This provides a **PSPACE**-upper bound for descriptive semantics, but leaves the lower bound open.

If the schema may even contain free assertions, reasoning becomes immediately more difficult. As proved in [116], it is sufficient to allow for the use of only class conjunction, universal quantification, and negation of class names (to express disjointness) to make reasoning **EXPTIME**-hard, i.e. as hard as in the most expressive languages and schemata considered in this thesis. The **EXPTIME**-hardness result for class consistency is established by polynomially reducing to this problem the problem of string-acceptance by a deterministic Turing Machine that works in exponential time. The TM is encoded in a schema in such a way that its computation corresponds to a model of the schema. The assumption that the machine works in exponential time makes it in fact possible to identify uniquely each configuration and each tape position using only a polynomial number of bits, and therefore of classes used to represent the bits.

The constructors for complex links, namely transitive closure, union and concatenation, are considered in [11], where it is shown that using such links the assertions of a free schema can be internalized in a formula. This result coincides in fact with a well known result about Propositional Dynamic Logics (see Theorem B.1.1), but has been proved independently without making use of the correspondence between Description Logics and PDLs discussed in Section B.2. Since the size of the formula is polynomial in the size of the schema it is obtained from, this result has also a direct impact on computational complexity. In fact, it shows that reasoning only on class expressions (i.e. without any schema) containing such complex link constructors is already **EXPTIME**-hard.

An immediate consequence of these results is that, if one wants to actually define classes in a schema (by necessary and sufficient conditions), one has immediately to give up worst case tractability of schema level reasoning, even for the most trivial class definition languages.

A question that arises naturally, is whether reasoning becomes easier, when we consider *primitive schemata*, which contain only primitive class specifications and no definitions. The importance of such schemata is motivated by the tight correspondence with database models, described in Chapter 3. They have been investigated only recently, however, and the impact on the complexity of reasoning of the assumption that the schema contains no definitions is not well understood. In particular, the known reasoning procedures cannot take advantage of the fact that the schema is primitive and treat it as an arbitrary schema possibly containing definitions. Also, the known lower bounds were the ones that follow trivially from the lower bounds established for reasoning on class expressions. Exceptions to this are the results established in [37], which we already mentioned. A consequence of the results in [40], which also deals explicitly with primitive schemata, is that for the language $\mathcal{L}^-$ class subsumption can be decided in polynomial time, even if the schema contains cycles[5].

[5]In fact, in [40] a more general language is considered, which, however, cannot be easily classified within the family of

The results we have established in this chapter show, however, that this is in fact the maximum expressivity we can achieve without giving up tractability. As soon as the class language allows one to define an inconsistent class expression (or in the schema we can express that two classes are disjoint), class consistency becomes intractable. The use of cycles, which are essential for modeling real world domains, and/or of disjunction increases the computational complexity even more. Qualified existential quantification on the other hand, does not seem to increase either expressivity or complexity of reasoning, at least for the more simple class definition languages.

The results about the intrinsic complexity of reasoning on primitive schemata are summarized in Table 4.1. All entries, except those in the first row, specify the computational complexity of class consistency. The entries in the first row specify the complexity of class subsumption, since in $\mathcal{L}^{-}$-schemata classes are always consistent. Each entry reports also a reference to the place (or places) where the result is achieved. The dictions **NP**, **coNP**, **PSPACE**, and **EXPTIME** mean that the corresponding problem is complete for the given class, while **PTIME** means that the problem is polynomially solvable. We have included for comparison also a column showing the complexity of reasoning on class expressions with respect to the empty schema.

$\mathcal{L}$-languages.

Chapter 5

Unrestricted Model Reasoning

In this chapter we discuss the issues related to reasoning on $\mathcal{L}$-schemata with respect to arbitrary models, i.e. models with a possibly infinite domain. The reasoning procedures we present take advantage of the strong similarity that exists between the interpretative structures of $\mathcal{L}$-schemata and labeled transition systems used in computer science to describe the behavior of program schemes. This similarity reflects in a correspondence between $\mathcal{L}$-schemata and modal logics of programs, which are formalisms specifically designed for reasoning about program schemes, and which are interpreted in terms of labeled transition systems (see [107, 150] for surveys). Such correspondence has been pointed out for the first time in [140]. It has successively been extended in [54, 55] to handle constructs such as number restrictions, that had not been considered in the context of modal logics of programs, and in [141, 55] to integrate the different types of semantics for cycles exploiting the relationship with μ-calculus [106].

A brief description of Propositional Dynamic Logics and of the main results for reasoning about them can be found in Appendix B. Appendix B describes also the above mentioned correspondence with $\mathcal{L}$-schemata. Section 5.1 describes how to exploit this correspondence to reason on primitive $\mathcal{LUNI}$-schemata. Finally, in Section 5.2 we show that for the language $\mathcal{LT}^-$, where the constructs of $\mathcal{LT}$ have been carefully tailored in order to avoid undecidability, reasoning is indeed decidable.

5.1 Unrestricted Model Reasoning on Primitive $\mathcal{LUNI}$-Schemata

In this section we present a technique for reasoning on primitive $\mathcal{LUNI}$-schemata with respect to unrestricted models. Table 5.1 recalls the constructors and assertions that are allowed in primitive $\mathcal{LUNI}$-schemata, and in the rest of the section we assume that all schemata are of this type.

The reasoning method exploits the correspondence between $\mathcal{L}$-languages and PDLs described in Section B.2, by reducing the problem of checking the consistency of a class in an $\mathcal{LUNI}$-schema to the problem of verifying the satisfiability of a formula of CPDL. However, because of the presence of number restrictions

Constructor Name	**Syntax**	**Semantics**
class name	C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
atomic negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$E_1 \sqcap E_2$	$E_1^{\mathcal{I}} \cap E_2^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$E_1 \sqcup E_2$	$E_1^{\mathcal{I}} \cup E_2^{\mathcal{I}}$
universal quantification	$\forall L.E$	$\{o \mid \forall o' : (o, o') \in L^{\mathcal{I}} \rightarrow o' \in E^{\mathcal{I}}\}$
(unqualified) exist. quantif.	$\exists L$	$\{o \mid \exists o' : (o, o') \in L^{\mathcal{I}}\}$
number restrictions ($\mathcal{N}$)	$\exists^{\geq m} L$	$\{o \mid \sharp\{o' \mid (o, o') \in L^{\mathcal{I}}\} \geq m\}$
	$\exists^{\leq n} L$	$\{o \mid \sharp\{o' \mid (o, o') \in L^{\mathcal{I}}\} \leq n\}$
attribute name	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
inverse attribute ($\mathcal{I}$)	A^-	$\{(o, o') \mid (o', o) \in A^{\mathcal{I}}\}$
primitive class specification	$C \dot{\preceq} E$	$C^{\mathcal{I}} \subseteq E^{\mathcal{I}}$

Table 5.1: Primitive $\mathcal{LUNI}$-schemata

in $\mathcal{LUNI}$ which have no correspondence in CPDL, we cannot exploit directly Theorem B.2.2. To overcome this problem we perform a (polynomial) transformation of the schema into one that does not contain any number restrictions and show that this transformation preserves class consistency. For the resulting schema we can exploit the correspondence with CPDL and obtain a decision procedure that works in worst case deterministic exponential time. For simplicity and uniformity, in the rest of the section we deal only with the modified version CAPDL of CPDL, obtained by considering as programs sequential automata instead of regular expression, as described in Section B.3.2.

5.1.1 Relaxation of a Schema

We define now a transformation on schemata which associates to each schema $\mathcal{S}$ a new schema $\mathcal{S}_{rel}$ in which the number restrictions that appear in the class expressions of $\mathcal{S}$ are relaxed.

Definition 5.1.1 The *relaxation* $\mathcal{S}_{rel} := (\mathcal{C}', \mathcal{A}', \mathcal{T}')$ of a schema $(\mathcal{C}, \mathcal{A}, \mathcal{T})$ is defined as follows:

- $\mathcal{C}' := \mathcal{C} \cup \mathcal{C}^{\geq} \cup \mathcal{C}^{\leq}$, where both unions are disjoint and

$$\begin{aligned} \mathcal{C}^{\geq} &:= \{C_{(\geq m\, L)} \mid \exists^{\geq m} L \text{ appears as a subexpression in an assertion of } \mathcal{T}\}, \\ \mathcal{C}^{\leq} &:= \{C_{(\leq n\, L)} \mid \exists^{\leq n} L \text{ appears as a subexpression in an assertion of } \mathcal{T}\}. \end{aligned}$$

- $\mathcal{A}' := \mathcal{A}$.
- $\mathcal{T}' := \mathcal{T}'' \cup \mathcal{T}_n \cup \mathcal{T}_e$, where
  - $\mathcal{T}''$ consists of the assertions of $\mathcal{T}$ in which each occurrence of a number restriction $\exists^{\geq m} L$ is replaced by $C_{(\geq m\, L)}$, and each occurrence of a number restriction $\exists^{\leq n} L$ is replaced by $\forall L.\bot$ if $n = 0$, and by $C_{(\leq n\, L)}$ if $n > 0$,
  - $\mathcal{T}_n$ contains an assertion $C_{(\geq m\, L)} \doteq \neg C_{(\leq n\, L)}$ for each pair of number restrictions $\exists^{\geq m} L$ and $\exists^{\leq n} L$ with $m > n$ present in $\mathcal{T}$;
  - $\mathcal{T}_e$ contains an assertion $C_{(\geq m\, L)} \doteq \exists L$ for each number restriction $\exists^{\geq m} L$ with $m > 0$ present in $\mathcal{T}$.

Lemma 5.1.2 *The relaxation $\mathcal{S}_{rel}$ of a schema $\mathcal{S}$ can be constructed in time linear in $|\mathcal{S}|$.*

Proof. Straightforward. □

The following proposition shows that the constraints that $\mathcal{S}_{rel}$ imposes on its models are indeed not stronger than the ones imposed by $\mathcal{S}$, since the additional assertions that have been added express only conditions that are implicit in the semantics of number restrictions.

Proposition 5.1.3 *Let $\mathcal{S}$ be a primitive $\mathcal{LUNI}$-schema and C be a class name of $\mathcal{S}$. If C is consistent in $\mathcal{S}$, then C is consistent in $\mathcal{S}_{rel}$.*

Proof. Let $\mathcal{I}$ be a model of $\mathcal{S}$ such that $C^{\mathcal{I}} \neq \emptyset$. We extend $\mathcal{I}$ to an interpretation of $\mathcal{S}_{rel}$ by interpreting the additional class names as follows:

$$\begin{aligned} C^{\mathcal{I}}_{(\geq m\, L)} &:= (\exists^{\geq m} L)^{\mathcal{I}} \\ C^{\mathcal{I}}_{(\leq n\, L)} &:= (\exists^{\leq n} L)^{\mathcal{I}}. \end{aligned}$$

The claim can then be shown easily by induction on the structure of class expressions that appear in the assertions of $\mathcal{S}$. □

Proposition 5.1.3 provides a necessary condition for the consistency of a class name in a schema which is stated in terms of its relaxation $\mathcal{S}_{rel}$. Since $\mathcal{S}_{rel}$ does not contain number restrictions, the consistency of a class in $\mathcal{S}_{rel}$ can be verified directly by making use of Theorem B.2.3 and applying reasoning techniques for CAPDL. We are going to show that this condition is also sufficient, i.e. that if a class is consistent in $\mathcal{S}_{rel}$ then it is also consistent in the original schema $\mathcal{S}$.

Let $f_{rel} := \delta^+(\mathcal{S}_{rel})$ be the CAPDL-formula corresponding to $\mathcal{S}_{rel}$. f_{rel} contains propositional letters $C_{(\geq m\, b)}$ and $C_{(\leq n\, b)}$ corresponding to the number restrictions in $\mathcal{S}$. We call such propositional letters

propositional number restrictions. Let $\mathcal{M} := (\mathcal{W}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \Pi^{\mathcal{M}})$ be a model of f_{rel} and $u \in \mathcal{W}^{\mathcal{M}}$. We say that the propositional number restriction $C_{(\geq m\, b)}$ is *numerically satisfied* in u, if $\sharp\{v \mid (u,v) \in \mathcal{R}^{\mathcal{M}}(b)\} \geq m$. Similarly, $C_{(\leq n\, b)}$ is numerically satisfied in u , if $\sharp\{v \mid (u,v) \in \mathcal{R}^{\mathcal{M}}(b)\} \leq n$. f_{rel} is *numerically satisfiable* if it admits a model $\mathcal{M}$ in which all propositional number restrictions are numerically satisfied in all states of $\mathcal{W}^{\mathcal{M}}$.

By Theorem B.2.2 models of $\mathcal{S}_{rel}$ and structures in which f_{rel} is valid are isomorphic. Recall that $\mathcal{S}_{rel}$ is obtained from $\mathcal{S}$ by replacing each number restriction with a new class name and by adding suitable assertions to the ones already present in $\mathcal{S}$. Therefore, any structure in which also all propositional number restrictions of f_{rel} are numerically satisfied is a model of $\mathcal{S}$. Moreover, since it is sufficient to consider connected structures (as shown below) , and for such structures validity and satisfiability of f_{rel} coincide, any model of f_{rel}[1] in which all propositional number restrictions are numerically satisfied is also a model of $\mathcal{S}$.

In the rest of the section we show how an arbitrary model $\mathcal{M}$ of f_{rel} can be transformed into one in which all propositional number restrictions are numerically satisfied. This is done in two steps, by first transforming $\mathcal{M}$ into a particular tree model and then by expanding this tree model in order to ensure that the propositional number restrictions are satisfied.

5.1.2 Two-Way Deterministic Tree Structures

By Theorem B.4.3 every satisfiable CAPDL-formula has a tree model. We show now that for the formulae resulting by applying the transformation δ^+ of Definition B.2.1 to primitive $\mathcal{LUNI}$ schemata an even stronger result holds. This is due to the simplified form that such formulae have.

Definition 5.1.4 A CAPDL-formula f is *primitive* if the following holds:

- In f negation is applied only to propositional letters.
- f contains no tests, i.e. the alphabets of the automata that define the programs that appear in f are constituted only by atomic programs[2].
- f contains only eventualities of the form $\langle b \rangle \top$ with b an atomic program. ∎

Lemma 5.1.5 *Let $\mathcal{S}$ be a primitive $\mathcal{LUNI}$-schema, $\mathcal{S}_{rel}$ its relaxation, and f_{rel} the* CAPDL*-formula obtained by applying transformation δ^+ to $\mathcal{S}_{rel}$. Then f_{rel} is a primitive* CAPDL*-formula.*

Proof. Straightforward by induction on the assertions of $\mathcal{S}_{rel}$. □

The fact that we deal only with primitive formulae is crucial for showing that the transformations we are going to perform on models preserve satisfiability. For proofs by induction we need to construct the closure of the formula we start with. If we constructed the Fischer-Ladner closure in the standard way of Definition B.4.1, not all formulae in the closure would be primitive. Therefore we define a restricted form of closure which contains only primitive CAPDL-formulae and which is nevertheless sufficient for our purposes.

Definition 5.1.6 The *restricted closure* $CL^-(f_0)$ of a primitive CPDL-formula f_0 is the least set Φ of formulae such that:

- $f_0 \in \Phi$.
- If $\neg p \in \Phi$ then $p \in \Phi$.
- If $p \in \Phi$ then $\neg p \in \Phi$.
- If $f \wedge f' \in \Phi$ then $f, f' \in \Phi$.
- If $f \vee f' \in \Phi$ then $f, f' \in \Phi$.
- If $[r]f \in \Phi$ then $f \in \Phi$.
- If $[r]f \in \Phi$, where $r = (\Sigma, S, \delta, s_0, F)$, then for all $s \in S$, $[r_s]f \in \Phi$, where $r_s := (\Sigma, S, \delta, s, F)$ is the automaton obtained from r by making s to the initial state. ∎

[1] A structure $\mathcal{M}$ is a model of a PDL-formula f, if $\mathcal{M}, u \models f$ for some $u \in \mathcal{W}^{\mathcal{M}}$.
[2] An atomic program is either a program name or a converse program name.

Definition 5.1.7 A structure $\mathcal{T} := (\mathcal{W}^{\mathcal{T}}, \mathcal{R}^{\mathcal{T}}, \Pi^{\mathcal{T}})$ is a *two-way deterministic tree structure* if it is a tree structure and for any program name a and states $x, y \in \mathcal{W}^{\mathcal{T}}$ the following holds: if $(x, y) \in \mathcal{R}^{\mathcal{T}}(a)$ then for any state $y' \neq y$, $(x, y') \notin \mathcal{R}^{\mathcal{T}}(a)$, and for any state $x' \neq x$, $(x', y) \notin \mathcal{R}^{\mathcal{T}}(a)$. ∎

Notice that any two-way deterministic tree structure $\mathcal{T}$ is an n-ary tree structure with $n \leq 2{\cdot}\sharp Prog$, since for each node x and each program name a there is at most one node y with $(x, y) \in \mathcal{R}^{\mathcal{T}}(a)$ and at most one node z with $(z, x) \in \mathcal{R}^{\mathcal{T}}(a)$.

The following lemma strengthens the tree model property for primitive CAPDL-formulae.

Lemma 5.1.8 *Let f be a satisfiable primitive* CAPDL*-formula. Then f has a two-way deterministic tree model $\mathcal{T}$, with $\mathcal{T}, \epsilon \models f$.*

Proof. Since f is satisfiable there is a structure $\mathcal{M} := (\mathcal{W}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \Pi^{\mathcal{M}})$ and a state $u \in \mathcal{W}^{\mathcal{M}}$ such that $\mathcal{M}, u \models f$. The proof consists of two parts: In part (1) we construct from $\mathcal{M}$ a two-way deterministic tree structure $\mathcal{T}$, and in part (2) we show that $\mathcal{T}$ is a model of f. We assume that the set of program names in f is $Prog := \{a_1, \ldots, a_k\}$.

(1) We define a partial mapping $\alpha : \{1, \ldots, 2{\cdot}k\}^* \to \mathcal{W}^{\mathcal{M}}$ by induction on the length of the words in $\{1, \ldots, 2{\cdot}k\}^*$. The relations $\mathcal{R}^{\mathcal{T}}(a_i)$ are defined in parallel with α. The set of states of $\mathcal{T}$ is given by $\mathcal{W}^{\mathcal{T}} := \{x \in \{1, \ldots, 2{\cdot}k\}^* \mid \alpha(x) \text{ is defined}\}$ and for all nodes $x \in \mathcal{W}^{\mathcal{T}}$, $\Pi^{\mathcal{T}}(x) := \Pi^{\mathcal{M}}(\alpha(x))$. For the root ϵ we define $\alpha(\epsilon) := u$. For the inductive step suppose that we have already considered every node of $\{1, \ldots, 2{\cdot}k\}^l$, where $0 \leq l \leq h$, and the nodes $xi1, xi2, \ldots, xi(j-1)$, for $xi \in \{1, \ldots, 2{\cdot}k\}^h$ with $\alpha(x_i)$ defined and $1 \leq j \leq 2{\cdot}k$. If j is odd, i.e. $j = 2{\cdot}g - 1$, where $1 \leq g \leq k$, let $v \in \mathcal{W}^{\mathcal{M}}$ be such that $(\alpha(xi), v) \in \mathcal{R}^{\mathcal{M}}(a_g)$. If such node v does not exist or if $(xi, x) \in \mathcal{R}^{\mathcal{T}}(a_g)$, then $\alpha(xij)$ stays undefined. Otherwise we define $\alpha(xij) := v$ and we put (xi, xij) in $\mathcal{R}^{\mathcal{T}}(a_g)$. If j is even, i.e. $j = 2{\cdot}g$, where $1 \leq g \leq k$, let $v \in \mathcal{W}^{\mathcal{M}}$ be such that $(v, \alpha(xi)) \in \mathcal{R}^{\mathcal{M}}(a_g)$. If such node v does not exist or if $(x, xi) \in \mathcal{R}^{\mathcal{T}}(a_g)$ then $\alpha(xij)$ stays undefined. Otherwise we define $\alpha(xij) := v$ and we put (xij, xi) in $\mathcal{R}^{\mathcal{T}}(a_g)$. By construction, $\mathcal{T}$ is a two-way deterministic tree structure.

(2) To prove that $\mathcal{T}, \epsilon \models f$, we show that for any formula $g \in CL^-(f)$ and any state $x \in \mathcal{W}^{\mathcal{T}}$, if $\mathcal{M}, \alpha(x) \models g$, then $\mathcal{T}, x \models g$. The proof is by induction on the structure of formulae.

- $g = p$, where p is a propositional letter. By construction of $\mathcal{T}$ we have that $\mathcal{T}, x \models p$ iff $\mathcal{M}, \alpha(x) \models p$.

- $g = \neg p$ where p is a propositional letter. Then $\mathcal{T}, x \models \neg g$ iff $\mathcal{T}, x \not\models g$ iff (by construction of $\mathcal{T}$) $\mathcal{M}, \alpha(x) \not\models g$ iff $\mathcal{M}, \alpha(x) \models \neg g$.

- $g = g_1 \wedge g_2$. Then $\mathcal{T}, x \models g_1 \wedge g_2$ iff $\mathcal{T}, x \models g_1$ and $\mathcal{T}, x \models g_2$ if (by induction hypothesis) $\mathcal{M}, \alpha(x) \models g_1$ and $\mathcal{M}, \alpha(x) \models g_2$ iff $\mathcal{M}, \alpha(x) \models g_1 \wedge g_2$.

- $g = g_1 \vee g_2$. Then $\mathcal{T}, x \models g_1 \vee g_2$ iff $\mathcal{T}, x \models g_1$ or $\mathcal{T}, x \models g_2$ if (by induction hypothesis) $\mathcal{M}, \alpha(x) \models g_1$ or $\mathcal{M}, \alpha(x) \models g_2$ iff $\mathcal{M}, \alpha(x) \models g_1 \vee g_2$.

- $g = \langle a_i \rangle \top$, where a_i is a program name. Since $\mathcal{M}, \alpha(x) \models \langle a_i \rangle \top$, there is a state $v \in \mathcal{W}^{\mathcal{M}}$ such that $(\alpha(x), v) \in \mathcal{R}^{\mathcal{M}}(a_i)$. The construction of $\mathcal{T}$ guarantees that there is some $y \in \mathcal{W}^{\mathcal{T}}$ such that $(x, y) \in \mathcal{R}^{\mathcal{T}}(a_i)$. In particular, either $x = z(2{\cdot}i)$ for some node $z \in \mathcal{W}^{\mathcal{T}}$ and $(x, z) \in \mathcal{R}^{\mathcal{T}}(a_i)$, or for the node $z' = x(2{\cdot}i - 1)$, $\alpha(z')$ is defined and $(x, z') \in \mathcal{R}^{\mathcal{T}}(a_i)$. Therefore $\mathcal{T}, x \models \langle a_i \rangle \top$.

- $g = \langle a_i^- \rangle \top$. We can proceed as in the previous case.

- $g = [r]g'$, where r is an arbitrary program. Let $b_1 b_2 \cdots b_l$ be any execution sequence of r such that there is a (unique) sequence of nodes $x_0, x_1, \ldots, x_l \in \mathcal{W}^{\mathcal{T}}$ with $x = x_0$ and $(x_{i-1}, x_i) \in \mathcal{R}^{\mathcal{T}}(b_i)$, for $1 \leq i \leq l$. It is sufficient to show that $\mathcal{T}, x_l \models g'$. By construction of $\mathcal{T}$, $\alpha(x_0), \alpha(x_1), \ldots, \alpha(x_l)$ are defined and $(\alpha(x_{i-1}), \alpha(x_i)) \in \mathcal{R}^{\mathcal{M}}(b_i)$, for $0 \leq i \leq l$. Since $\mathcal{M}, \alpha(x_0) \models [r]g'$, we have that $\mathcal{M}, \alpha(x_i) \models g'$, and by induction hypothesis $\mathcal{T}, x_i \models g'$. Thus $\mathcal{T}, x \models [r]g'$.

□

5.1.3 Expansion of Two Way Deterministic Tree Structures

We show now how any two-way deterministic tree model $\mathcal{T}$ of a primitive CPDL-formula $f_{rel} := \delta^+(\mathcal{S}_{rel})$ corresponding to the expansion of a primitive $\mathcal{LUNI}$-schema $\mathcal{S}$ can be expanded to a model $\mathcal{M}$ in which all propositional number restrictions present in the formula are numerically satisfied. First of all notice that all propositional number restrictions of the form $C_{(\leq n\, b)}$ are already numerically satisfied in all states of $\mathcal{T}$. This is because in a two-way deterministic tree model by definition for each state x and atomic program b there is at most one other state connected to x through b. Notice also that for the same reason no propositional number restriction of the form $C_{(\geq m\, b)}$ with $m > 1$ is numerically satisfied in $\mathcal{T}$.

From now on let $Prog := \{a_1, \ldots, a_k\}$ be the set of program names that appear in f_{rel}. For a structure $\mathcal{M} := (\mathcal{W}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \Pi^{\mathcal{M}})$, an atomic program $b \in Prog \cup Prog^-$ and a state $x \in \mathcal{W}^{\mathcal{M}}$, let

$$\begin{aligned} m_{max}(\mathcal{M}, x, b) &:= \max\{m \mid C_{(\geq m\, b)} \in \Pi^{\mathcal{M}}(x)\} \\ n_{min}(\mathcal{M}, x, b) &:= \min\{n \mid C_{(\leq n\, b)} \in \Pi^{\mathcal{M}}(x)\}. \end{aligned}$$

If $\Pi^{\mathcal{M}}(x)$ does not contain any propositional number restriction $C_{(\geq m\, b)}$, then $m_{max}(x, b) := 0$. Similarly, if $\Pi^{\mathcal{M}}(x)$ does not contain any propositional number restriction $C_{(\leq n\, b)}$, then $n_{min}(x, b) := \infty$.

Lemma 5.1.9 *Let $\mathcal{T}$ be a tree model of f_{rel}. Then for any state $x \in \mathcal{W}^{\mathcal{T}}$ and any atomic program $b \in Prog \cup Prog^-$ the following holds*

1. *$m_{max}(\mathcal{T}, x, b) \leq n_{min}(\mathcal{T}, x, b)$.*
2. *If $m_{max}(\mathcal{T}, x, b) \geq 1$ then there is a state $y \in \mathcal{W}^{\mathcal{T}}$ such that $(x, y) \in \mathcal{R}^{\mathcal{T}}(b)$.*

Proof. For (1) suppose by contradiction that $m_{max}(\mathcal{T}, x, b) > n_{min}(\mathcal{T}, x, b)$ for some state $x \in \mathcal{W}^{\mathcal{T}}$ and some atomic program $b \in Prog \cup Prog^-$. Then $\Pi^{\mathcal{T}}(x)$ contains two propositional number restrictions $C_{(\geq m_0\, b)}$ and $C_{(\leq n_0\, b)}$, where $m_0 := m_{max}(\mathcal{T}, x, b)$ and $n_0 := n_{min}(\mathcal{T}, x, b)$. By definition of m_{max} and n_{min}, $\mathcal{S}_{rel}$ contains an assertion $C_{(\geq m_0\, b)} \dot{\preceq} \neg C_{(\leq n_0\, b)}$. Since $\mathcal{T}$ is a model of f_{rel}, there is a state $x_0 \in \mathcal{W}^{\mathcal{T}}$ such that $\mathcal{T}, x_0 \models [r_C](C_{(\geq m_0\, b)} \rightarrow \neg C_{(\leq n_0\, b)})$, where r_C is the automaton that accepts $(a_1 \cup \cdots \cup a_k \cup a_1^- \cup \cdots \cup a_k^-)^*$. Since $\mathcal{T}$ is connected there are $l \geq 0$, states $x_1, \ldots, x_l$, and atomic programs $b_1, \ldots, b_l$ such that $x_l = x$ and $(x_{i-1}, x_i) \in \mathcal{R}^{\mathcal{T}}(b_i)$, for $1 \leq i \leq l$. The word $b_1 b_2 \cdots b_l$ is an execution sequence of r_C, and therefore $\mathcal{T}, x \models (C_{(\geq m_0\, b)} \rightarrow \neg C_{(\leq n_0\, b)})$, which gives rise to a contradiction.

For (2) suppose by contradiction that $m_0 := m_{max}(\mathcal{T}, x, b) > 0$ for some state $x \in \mathcal{W}^{\mathcal{T}}$ and some atomic program $b \in Prog \cup Prog^-$, and that there is no state $y \in \mathcal{W}^{\mathcal{T}}$ such that $(x, y) \in \mathcal{R}^{\mathcal{T}}(b)$. Since $m_0 > 0$, by definition of m_{max}, $\Pi^{\mathcal{T}}(x)$ contains a propositional number restriction $C_{(\geq m_0\, b)}$ and since $\mathcal{T}$ is a model of f_{rel}, there is a state $x_0 \in \mathcal{W}^{\mathcal{T}}$ such that $\mathcal{T}, x_0 \models [r_C](C_{(\geq m_0\, b)} \rightarrow \langle b \rangle \top)$, where r_C is as above. By reasoning as in the previous case we get a contradiction from the fact that both $\mathcal{T}, x \models C_{(\geq m_0\, b)}$ and $\mathcal{T}, x \models [b]\bot$. □

In order to obtain from a two-way deterministic tree model $\mathcal{T}$ of f_{rel} a model $\mathcal{M}$ in which all propositional number restrictions are numerically satisfied, it is sufficient to ensure that those of the type $C_{(\geq m\, b)}$, with $m > 1$ are satisfied. Part (1) of Lemma 5.1.9 guarantees that by modifying $\mathcal{T}$ in order to numerically satisfy $C_{(\geq m\, b)}$ we introduce no contradictions with propositional number restrictions of the type $C_{(\leq n\, b)}$. Part (2) of Lemma 5.1.9 guarantees that whenever we have to numerically satisfy $C_{(\geq m\, b)}$ in a node x, there is already a node y in $\mathcal{T}$ such that $(x, y) \in \mathcal{R}^{\mathcal{T}}(b)$. Therefore we can construct a model $\mathcal{M}$ of f_{rel} by starting from a two-way deterministic tree model $\mathcal{T}$ and duplicating suitable subtrees connected to each node.

Let $\mathcal{T} := (\mathcal{W}^{\mathcal{T}}, \mathcal{R}^{\mathcal{T}}, \Pi^{\mathcal{T}})$ be a two-way deterministic tree structure obtained from a model of f_{rel} by applying mapping α as defined in the proof of Lemma 5.1.8. For every node $xi \in \mathcal{W}^{\mathcal{T}}$ of such a tree structure $\mathcal{T}$ the following holds:

1. If i is odd, i.e. $i = 2{\cdot}g - 1$, where $1 \leq g \leq k$, then $(x, xi) \in \mathcal{R}^{\mathcal{T}}(a_g)$ and $xi(i+1) \notin \mathcal{W}^{\mathcal{T}}$.
2. If i is even, i.e. $i = 2{\cdot}g$, where $1 \leq g \leq k$, then $(xi, x) \in \mathcal{R}^{\mathcal{T}}(a_g)$ and $xi(i-1) \notin \mathcal{W}^{\mathcal{T}}$.

Such property allows us to define the following transformations on tree structures. Together with the transformations we define also mappings from nodes of the tree generated by the transformation to nodes of the original tree. Let m be a positive integer, $\mathcal{T}_j$ be a $2{\cdot}k_j$-ary two-way deterministic tree structure as above, for $1 \leq j \leq m$, and $x, y \in \mathcal{W}^{\mathcal{T}_1}$ two nodes that are neighbors.

- $\mathcal{T}' := cut(x, y, \mathcal{T}_1)$ is the tree structure obtained from $\mathcal{T}_1$ by making x to be the new root of the tree and eliminating from the resulting tree all subtrees except the one rooted at y. In this way we ensure also that the root of $\mathcal{T}'$ has at most one successor. In order to define a suitable mapping $\gamma_{cut}(x, y, w)$ between nodes, we distinguish two cases:

  If y is the i-successor of x, i.e. $y = xi$ where $1 \leq i \leq 2{\cdot}k_1$, then

  $$\gamma_{cut}(x, y, w) := \begin{cases} x, & \text{if } w = \epsilon. \\ xw, & \text{if } w = iw'. \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

  If x is the i-successor of y in $\mathcal{T}_1$, i.e. $x = yi$, the formal definitions of $\gamma_{cut}(x, y, w)$ and of $\mathcal{T}'$ are complicated by the fact that we have to invert the predecessor relation for the nodes on the path from the root to x. In order to do this without introducing conflicts, properties (1) and (2) established above are crucial. For a word $w := i_1 \cdots i_h \in \{1, \ldots, 2{\cdot}k_1\}^*$ we define $\tilde{w} := \tilde{\imath}_h \cdots \tilde{\imath}_1$, where $\tilde{\imath}_j := i_j + 1$ if i_j is odd, and $\tilde{\imath}_j := i_j - 1$ if i_j is even. Let $w = vw'$ and $\tilde{x} = vx'$, where v is the common prefix of w and $\tilde{x}$. Then

  $$\gamma_{cut}(x, y, w) := \begin{cases} x, & \text{if } w = \epsilon. \\ \tilde{x}'w', & \text{if } |v| \geq 1. \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

  In both cases, $\mathcal{T}'$ is a $2{\cdot}k_1$-ary tree structure defined as follows:

  - $\mathcal{W}^{\mathcal{T}'} := \{w \in \{1, \ldots, 2{\cdot}k_1\}^* \mid \gamma_{cut}(x, y, w) \in \mathcal{W}^{\mathcal{T}_1}\}$.
  - $\Pi^{\mathcal{T}'}(w) := \Pi^{\mathcal{T}_1}(\gamma_{cut}(x, y, w))$.
  - $\mathcal{R}^{\mathcal{T}'}(a) := \{(w, w') \mid (\gamma_{cut}(x, y, w), \gamma_{cut}(x, y, w')) \in \mathcal{R}^{\mathcal{T}_1}(a)\}$.

- $\mathcal{T}' := copy(m, \mathcal{T}_1)$ is the tree structure obtained from $\mathcal{T}$ by substituting for each node i at depth 1 the whole subtree starting at i with a subtree starting at $m{\cdot}i$. We define the following mapping between nodes:

  $$\gamma_{copy}(m, w) := \begin{cases} \epsilon, & \text{if } w = \epsilon. \\ iw', & \text{if } w = (m{\cdot}i)w'. \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

  Formally, $\mathcal{T}'$ is the $2{\cdot}k{\cdot}m$-ary tree structure defined as follows:

  - $\mathcal{W}^{\mathcal{T}'} := \{w \in \{1, \ldots, 2{\cdot}k_1{\cdot}m\}^* \mid \gamma_{copy}(m, w) \in \mathcal{W}^{\mathcal{T}_1}\}$.
  - $\Pi^{\mathcal{T}'}(w) := \Pi^{\mathcal{T}_1}(\gamma_{copy}(m, w))$.
  - $\mathcal{R}^{\mathcal{T}'}(a) := \{(w, w') \mid (\gamma_{copy}(m, w), \gamma_{copy}(m, w')) \in \mathcal{R}^{\mathcal{T}_1}(a)\}$.

- $\mathcal{T}' := app(x, (\mathcal{T}_2, \ldots, \mathcal{T}_m), \mathcal{T}_1)$ is the tree structure obtained by appending $\mathcal{T}_2, \ldots, \mathcal{T}_m$ at node x of $\mathcal{T}_1$. We define the following mapping between nodes:

  $$\gamma_{app}(x, w) := \begin{cases} w', & \text{if } w = xw'. \\ w, & \text{otherwise.} \end{cases}$$

  Formally, $\mathcal{T}'$ is the $2{\cdot}\sum_{1 \leq i \leq m} k_i$-ary tree structure defined as follows:

  - $\mathcal{W}^{\mathcal{T}'} := \mathcal{W}^{\mathcal{T}_1} \cup \bigcup_{1 \leq i \leq m} \{xw \mid w \in \mathcal{W}^{\mathcal{T}_i}\}$.
  - $\Pi^{\mathcal{T}'}(w) := \begin{cases} \Pi^{\mathcal{T}_j}(\gamma_{app}(x, w)), & \text{if } j \text{ is the least } i,\ 2 \leq i \leq m, \\ & \text{with } \gamma_{app}(x, w) \in \mathcal{W}^{\mathcal{T}_i}. \\ \Pi^{\mathcal{T}_1}(w), & \text{otherwise.} \end{cases}$
  - $\mathcal{R}^{\mathcal{T}'}(a) := \mathcal{R}^{\mathcal{T}_1}(a) \cup \{(w, w') \mid (\gamma_{app}(x, w), \gamma_{app}(x, w')) \in \mathcal{R}^{\mathcal{T}_i}(a)$ for some $i, 2 \leq i \leq m\}$.

  Notice that according to this definition $app(x, (\mathcal{T}_2, \ldots, \mathcal{T}_m), \mathcal{T}_1)$ is guaranteed to be a tree structure only if the set of nodes of the trees $\mathcal{T}_i$ are pairwise disjoint and moreover no tree $\mathcal{T}_i$ interferes with the subtree of $\mathcal{T}_1$ rooted at x. That this condition is verified for the trees we consider is ensured by the construction of $\mathcal{M}$. Moreover, if for all trees $\mathcal{T}_i$ the root has only one successor, then $\mathcal{T}'$ is a $2{\cdot}\max\{k_1 + m, k_2, \ldots, k_m\}$-ary tree structure.

Let now m_0 be the maximum number that appears in a propositional number restriction of the form $C_{(\geq m\, r)}$, i.e.

$$m_0 := \max\{m \mid \exists b \in \mathit{Prog} \cup \mathit{Prog}^- : C_{(\geq m\, b)} \text{ appears in } f_{rel}\}.$$

The transformations defined above are used to construct, starting from $\mathcal{T}$, a $M := 2{\cdot}k{\cdot}m_0$-ary tree structure $\mathcal{M}$, called the *expansion* of $\mathcal{T}$, in which all propositional number restrictions are numerically satisfied.

Construction 5.1.10 The construction of the expansion $\mathcal{M}$ of $\mathcal{T}$ proceeds by induction on the length of the words in $\{1,\ldots,M\}^*$ and on the number of atomic programs. In parallel with the construction of $\mathcal{M}$ we define two mappings $\beta\colon \mathcal{W}^{\mathcal{M}} \to \mathcal{W}^{\mathcal{T}}$ and $\gamma\colon \mathcal{W}^{\mathcal{M}} \to \mathcal{W}^{\mathcal{M}}$. β assigns to each node of $\mathcal{M}$ a node of $\mathcal{T}$ that represents its first copy and γ assigns to each node the last node it is a copy from. We call the application of each step of the construction to a node x and an atomic program b_h the *expansion of x with respect to b_h*.

For the base step we put $\mathcal{M}$ equal to $\mathcal{T}$ and for every node $x \in \mathcal{W}^{\mathcal{M}}$ we define $\beta(x) := x$ and $\gamma(x) := x$.

For the inductive step suppose that we have already expanded the following nodes:

- All nodes in $\{1,\ldots,M\}^{l-1}$ with respect to all atomic programs.
- For some $x \in \{1,\ldots,M\}^{l-1}$ and some $i \in \{1,\ldots,M\}$, all nodes in $\{1,\ldots,M\}^{l}$ that lexicographically precede xi with respect to all atomic programs.
- The node xi with respect to the atomic programs $b_1,\ldots,b_{h-1}$.

Let $m := m_{max}(\mathcal{M}, xi, b_h)$. We describe now the expansion of node xi with respect to b_h. If $m \geq 1$, by part (2) of Lemma 5.1.9 there is a node $y \in \mathcal{W}^{\mathcal{M}}$ such that $(xi, y) \in \mathcal{R}^{\mathcal{M}}(b_h)$. If $m = 1$, or if $m > 1$ and there is more than one node $y \in \mathcal{W}^{\mathcal{M}}$ such that $(xi, y) \in \mathcal{R}^{\mathcal{M}}(b_h)$[3], we are done and we do not modify $\mathcal{M}$. Otherwise $m > 1$ and there is exactly one node $y \in \mathcal{W}^{\mathcal{M}}$ such that $(xi, y) \in \mathcal{R}^{\mathcal{M}}(b_h)$.

Let $\mathcal{T}_j := \mathit{copy}(j, \mathit{cut}(xi, y, \mathcal{M}))$, $2 \leq j \leq m$. For any node $z \in \mathcal{W}^{\mathcal{T}_j}$, let $\gamma'(z) := \gamma_{cut}(xi, y, \gamma_{copy}(m, z))$ be the original node in $\mathcal{M}$ of which z is a copy. Then we append the tree structures $\mathcal{T}_2,\ldots,\mathcal{T}_m$ to node xi, which formally corresponds to replacing $\mathcal{M}$ with $\mathcal{M}' := \mathit{app}(xi, (\mathcal{T}_2,\ldots,\mathcal{T}_m), \mathcal{M})$. Finally we put for any node $z \in \mathcal{W}^{\mathcal{M}'} \setminus \mathcal{W}^{\mathcal{M}}$, $\beta(z) := \beta(z')$ and $\gamma(z) := z'$, where $z' := \gamma'(\gamma_{app}(xi, z))$. ∎

We observe that by construction the following holds: (1) Since each tree $\mathcal{T}_j$ is obtained as a copy with a different coefficient, the trees $\mathcal{T}_j$ do not interfere with each other and with the subtree of $\mathcal{M}$ rooted at xi. (2) Since each tree $\mathcal{T}_j$ has at most one successor of the root, and since the expansion had not already been applied to node xi, the resulting structure $\mathcal{M}$ is still an M-ary tree structure. (3) $\beta(x) \in \mathcal{W}^{\mathcal{T}}$ for all $x \in \mathcal{W}^{\mathcal{M}}$, and $\gamma(x)$ is the node from which x was generated in an expansion step. (4) $\gamma(x)$ precedes x in $\mathcal{M}$ if we visit $\mathcal{M}$ breadth-first.

Observations (1) and (2) allow us to proceed with the inductive construction, while observations (3) and (4) are used below. The following two lemmata characterize the properties of tree structures that are obtained by the previous construction.

Lemma 5.1.11 *Let $\mathcal{M}$ be the expansion of a two-way deterministic tree model $\mathcal{T}$ of f_{rel}, obtained according to Construction 5.1.10 and let $u \in \mathcal{W}^{\mathcal{M}}$ and $f \in CL^-(f_{rel})$. Then $\mathcal{M}, u \models f$ if and only if $\mathcal{T}, \beta(u) \models f$.*

Proof. The proof is by induction on the structure of formulae.

- $f = p$, where p is a propositional letter. By construction of $\mathcal{M}$ we have that $\mathcal{M}, u \models p$ iff $\mathcal{T}, \beta(u) \models p$.
- $f = \neg p$ where p is a propositional letter. $\mathcal{M}, u \models \neg p$ iff $\mathcal{M}, u \not\models p$ iff (by induction hypothesis) $\mathcal{T}, \beta(u) \not\models p$ iff $\mathcal{T}, \beta(u) \models \neg p$.
- $f = f' \wedge f''$. $\mathcal{M}, u \models f' \wedge f''$ iff $\mathcal{M}, u \models f'$ and $\mathcal{M}, u \models f''$ iff (by induction hypothesis) $\mathcal{T}, \beta(u) \models f'$ and $\mathcal{T}, \beta(u) \models f''$ iff $\mathcal{T}, \beta(u) \models f' \wedge f''$
- $f = f' \vee f''$. $\mathcal{M}, u \models f' \vee f''$ iff $\mathcal{M}, u \models f'$ or $\mathcal{M}, u \models f''$ iff (by induction hypothesis) $\mathcal{T}, \beta(u) \models f'$ or $\mathcal{T}, \beta(u) \models f''$ iff $\mathcal{T}, \beta(u) \models f' \vee f''$

[3] This means that xi is actually a copy of a node that has already been expanded with respect to b_h.

- $f = \langle a_h \rangle \top$. "$\Rightarrow$" Suppose that $\mathcal{M}, u \models \langle a_h \rangle \top$. Then there is a node $v \in \mathcal{W}^{\mathcal{M}}$ such that $(u, v) \in \Pi^{\mathcal{M}}(a_h)$. By construction of $\mathcal{M}$ we have that $(\beta(u), \beta(v)) \in \mathcal{R}^{\mathcal{T}}(a_h)$ and therefore $\mathcal{T}, \beta(u) \models \langle a_h \rangle \top$.

  "$\Leftarrow$" Suppose that $\mathcal{T}, \beta(u) \models \langle a_h \rangle \top$. Then there is a node $v' \in \mathcal{W}^{\mathcal{T}}$ such that $(\beta(u), v') \in \mathcal{R}^{\mathcal{T}}(a_h)$. We show by induction on the number of expansion steps in the construction of $\mathcal{M}$, that there is a node $v \in \mathcal{W}^{\mathcal{M}}$ such that $(u, v) \in \mathcal{R}^{\mathcal{M}}(a_h)$. If $u \in \mathcal{W}^{\mathcal{T}}$ then $u = \beta(u)$ and $(u, v') \in \mathcal{R}^{\mathcal{M}}(a_h)$, since $\mathcal{R}^{\mathcal{T}}(a) \subseteq \mathcal{R}^{\mathcal{M}}(a)$ for all program names a. Otherwise, u has been added to $\mathcal{M}$ in some expansion step. Let $\mathcal{M}'$ be the tree structure immediately before performing this expansion step, and let the expansion step be on node x with respect to a_i and defined as follows: For $\mathcal{T}_j := copy(j, cut(x, y, \mathcal{M}'))$, where $2 \leq j \leq m := m_{max}(\mathcal{M}', x, a_i)$ let $\mathcal{M}'' := app(x, (\mathcal{T}_2, \ldots, \mathcal{T}_m), \mathcal{M}')$. Then $u \in \mathcal{W}^{\mathcal{M}''} \setminus \mathcal{W}^{\mathcal{M}'}$ and $u' := \gamma(u)$, is by construction a node of $\mathcal{M}'$. Again by construction $\beta(u') = \beta(u)$, and it follows by induction hypothesis that $\mathcal{M}', u' \models \langle a_h \rangle \top$. Therefore there is a node $v' \in \mathcal{W}^{\mathcal{M}'}$ such that $(u', v') \in \mathcal{R}^{\mathcal{M}'}(a_h)$. Since $u \notin \mathcal{W}^{\mathcal{M}'}$, we have $u_1 := \gamma_{app}(x, u) \in \mathcal{W}^{\mathcal{T}_j}$ for some j, $2 \leq j \leq m$. Consider the node $v_1 \in \mathcal{W}^{\mathcal{T}_j}$ such that $v' = \gamma_{cut}(x, y, \gamma_{copy}(j, v_1))$. Such a node necessarily exists by definition of *cut* and *copy*. In fact, either $v' = x$, i.e. $v_1 = \epsilon$, in which case $a_i = a_h$, $u' = y$, and by construction of $\mathcal{M}''$, $(u, x) \in \mathcal{R}^{\mathcal{M}''}(a_h)$. Or $v' \in \mathcal{W}^{\mathcal{M}''} \setminus \mathcal{W}^{\mathcal{M}'}$ and again by definition of *cut* and *copy* $(u_1, v_1) \in \mathcal{R}^{\mathcal{T}_j}(a_h)$. But then, by definition of *app* there is a node $v \in \mathcal{W}^{\mathcal{M}''} \setminus \mathcal{W}^{\mathcal{M}'}$ such that $\gamma_{app}(x, v) = v_1$, and $(u, v) \in \mathcal{R}^{\mathcal{M}''}(a_h)$.

- $f = \langle a_h^- \rangle \top$. We can proceed as in the previous case.

- $f = [r]f'$. "$\Rightarrow$" Suppose that $\mathcal{M}, u \models [r]f'$, and let $b_1 b_2 \ldots b_k$ be an arbitrary execution sequence of r. Let $x_0, x_1, \ldots, x_k$ be a sequence of nodes of $\mathcal{T}$ such that $x_0 = \beta(u)$ and $(x_{i-1}, x_i) \in \mathcal{R}^{\mathcal{T}}(b_i)$, for $1 \leq i \leq k$. If such sequence of nodes does not exist, then we are done. If such sequence of nodes exists, then it is necessarily unique since $\mathcal{T}$ is a two-way deterministic tree structure. Therefore it is sufficient to prove that $\mathcal{T}, x_k \models f'$. We do this by induction on k.

  - If $k = 0$ then $\mathcal{M}, u \models f'$. By induction on the structure of formulae $\mathcal{T}, \beta(u) \models f'$.
  - If $k \geq 1$, then $\mathcal{T}, x_0 \models \langle b_1 \rangle \top$. By the previous case, since $\beta(u) = x_0$, $\mathcal{M}, u \models \langle b_1 \rangle \top$ and there is a node $u_1 \in \mathcal{W}^{\mathcal{M}}$ such that $(u, u_1) \in \mathcal{R}^{\mathcal{M}}(b_1)$. Let $r := (\Sigma, S, \delta, s_0, F)$ and consider the program $r_s := (\Sigma, S, \delta, s, F)$ where $s := \delta(s_0, b_1)$. Then $\mathcal{M}, u_1 \models [r_s]f'$, and $b_2 \ldots b_k$ is an execution sequence of r_s. By construction $(\beta(u), \beta(u_1)) \in \mathcal{R}^{\mathcal{T}}(b_i)$, and since x_1 is the only node in $\mathcal{W}^{\mathcal{T}}$ such that $(x_0, x_1) \in \mathcal{R}^{\mathcal{T}}(b_i)$, we have that $x_1 = \beta(u_1)$. Since $\mathcal{M}, u_1 \models [r_s]f'$, $b_2 \ldots b_k$ is an execution sequence of r_s of length $k - 1$, and $x_1, \ldots, x_k$ is a sequence of nodes of $\mathcal{T}$ such that $x_1 = \beta(u_1)$ and $(x_{i-1}, x_i) \in \mathcal{R}^{\mathcal{T}}(b_i)$, for $2 \leq i \leq k$, it follows by induction on k that $\mathcal{T}, x_k \models f'$.

  "$\Leftarrow$" Suppose that $\mathcal{T}, \beta(u) \models [r]f'$, and let $b_1 b_2 \ldots b_k$ be an arbitrary execution sequence of r. Let $u_0, u_1, \ldots, u_k$ be an arbitrary sequence of nodes of $\mathcal{M}$ such that $u_0 = u$ and $(u_{i-1}, u_i) \in \mathcal{R}^{\mathcal{M}}(b_i)$, for $1 \leq i \leq k$. If such sequence of nodes does not exist, then we are done. If such sequence of nodes exists, then we prove by induction on k that $\mathcal{M}, u_k \models f'$.

  - If $k = 0$ then $\mathcal{T}, \beta(u) \models f'$. By induction on the structure of formulae $\mathcal{M}, u \models f'$.
  - If $k \geq 1$, then $\mathcal{M}, u_0 \models \langle b_1 \rangle \top$. Let $x_0 := \beta(u_0)$. By the previous case, $\mathcal{T}, x_0 \models \langle b_1 \rangle \top$ and there is a node $x_1 \in \mathcal{W}^{\mathcal{T}}$ such that $(x_0, x_1) \in \mathcal{R}^{\mathcal{T}}(b_1)$. Let $r := (\Sigma, S, \delta, s_0, F)$ and consider the program $r_s := (\Sigma, S, \delta, s, F)$ where $s := \delta(s_0, b_1)$. Then $\mathcal{T}, x_1 \models [r_s]f'$, and $b_2 \ldots b_k$ is an execution sequence of r_s. By construction $(\beta(u_0), \beta(u_1)) \in \mathcal{R}^{\mathcal{T}}(b_i)$, and since x_1 is the only node in $\mathcal{W}^{\mathcal{T}}$ such that $(x_0, x_1) \in \mathcal{R}^{\mathcal{T}}(b_i)$, we have that $x_1 = \beta(u_1)$. Since $\mathcal{T}, x_1 \models [r_s]f'$, $b_2 \ldots b_k$ is an execution sequence of r_s of length $k - 1$, and $u_1, \ldots, u_k$ is a sequence of nodes of $\mathcal{M}$ such that $x_1 = \beta(u_1)$ and $(u_{i-1}, u_i) \in \mathcal{R}^{\mathcal{M}}(b_i)$, for $2 \leq i \leq k$, it follows by induction on k that $\mathcal{M}, u_k \models f'$.

□

Lemma 5.1.12 *Let $\mathcal{M}$ be the expansion of a two-way deterministic tree model $\mathcal{T}$ of f_{rel}. Then all propositional number restrictions are numerically satisfied in all nodes of $\mathcal{M}$.*

Proof. With respect to propositional number restrictions of the form $C_{(\leq n\, b)}$, we observe that they are numerically satisfied in $\mathcal{M}$ in all nodes that have not been expanded. Note also that in the construction of $\mathcal{M}$ each node of $\mathcal{M}$ has been expanded at most once with respect to each atomic program. Therefore,

let u be a node of $\mathcal{M}$, b_h be a basic program of f_{rel}, $\mathcal{M}'$ be the structure before expanding node u with respect to b_h, $C_{(\geq m\, b_h)}$ be a propositional number restriction not satisfied in s in $\mathcal{M}'$, and $\mathcal{M}''$ be the structure obtained by expanding u with respect to b_h. Part (2) of Lemma 5.1.9 guarantees that there is at least one b_h-neighbor of u in $\mathcal{T}$ and therefore also in $\mathcal{M}'$. If there is exactly one such neighbor we expand u with respect to b_h. Since $m_{max}(\mathcal{T}, u, b_h) \geq m$, by expanding u with respect to b_h we force $C_{(\geq m\, b_h)}$ to be numerically satisfied in $\mathcal{M}''$ and therefore also in $\mathcal{M}$. Part (1) of Lemma 5.1.9 guarantees that u still numerically satisfies in $\mathcal{M}''$, and therefore also in $\mathcal{M}$, all propositional number restrictions of the type $C_{(\leq n\, b_h)}$. If we assume that there is more than one b_h-neighbor of u, then u is the copy of the node $\gamma(u)$. Since $\gamma(u)$ precedes u in $\mathcal{M}_h$ it has already been expanded with respect to b_h. Since $\Pi^{\mathcal{M}'}(u) = \Pi^{\mathcal{M}'}(\gamma(u))$, we have that $m_{max}(\mathcal{M}', u, b_h) = m_{max}(\mathcal{M}', \gamma(u), b_h)$ and $\gamma(u)$ numerically satisfies $C_{(\geq m\, b_h)}$. By construction u and $\gamma(u)$ have the same number of b_h-neighbors, and we get a contradiction to the fact that $C_{(\geq m\, R)}$ is not numerically satisfied in u. □

5.1.4 Upper Bounds for Unrestricted Model Reasoning

The following lemma summarizes the results of this section.

Lemma 5.1.13 *Let $f_{rel} := \delta^+(\mathcal{S}_{rel})$ be a primitive* CPDL*-formula obtained from the relaxation $\mathcal{S}_{rel}$ of a primitive $\mathcal{LUNI}$-schema. If f_{rel} is satisfiable then it is numerically satisfiable.*

Proof. Since f_{rel} is satisfiable, by Lemma 5.1.8 f_{rel} has a two-way deterministic tree model $\mathcal{T}$. Let $\mathcal{M}$ be the expansion of $\mathcal{T}$ obtained by Construction 5.1.10. By Lemma 5.1.11 $\mathcal{M}$ is a model of f_{rel} and by Lemma 5.1.12 all propositional number restrictions are numerically satisfied in $\mathcal{M}$. Therefore f_{rel} is numerically satisfiable. □

Proposition 5.1.14 *Let $\mathcal{S}$ be a primitive $\mathcal{LUNI}$-schema and C a class name of $\mathcal{S}$. Then C is consistent in $\mathcal{S}$ only if C is consistent in $\mathcal{S}_{rel}$.*

Proof. Let C be consistent in $\mathcal{S}_{rel}$. By Theorem B.2.3, $\delta^+(\mathcal{S}_{rel}) \wedge \delta(C)$ is satisfiable. By Lemma 5.1.13, it is numerically satisfiable. More precisely, it admits a connected model $\mathcal{M}$ in which all propositional number restrictions are numerically satisfied in all states of $\mathcal{M}$. By Theorem B.2.2, $\delta^+(\mathcal{S}_{rel})$ is valid in $\mathcal{M}$ and $\mathcal{M}$ is isomorphic to a model $\mathcal{I}$ of $\mathcal{S}_{rel}$. Since $\mathcal{M}$ is also a model of $\delta(C)$, C is consistent in $\mathcal{I}$, and since all propositional number restrictions of $\delta^+(\mathcal{S}_{rel})$ are numerically satisfied in all states of $\mathcal{M}$, we have that $\mathcal{I}$ is in fact a model of $\mathcal{S}$. □

We obtain as an immediate consequence the following upper bound for unrestricted class consistency in primitive $\mathcal{LUNI}$-schemata

Theorem 5.1.15 *Unrestricted class consistency $\mathcal{S} \not\models_u E \equiv \bot$ in primitive $\mathcal{LUNI}$-schemata can be decided in worst case deterministic exponential time in $|\mathcal{S}| + |E|$.*

Proof. If E is not a class name, by Proposition 2.3.5, E is consistent in $\mathcal{S}$ if and only if C is consistent in $\mathcal{S}' := (\mathcal{C}', \mathcal{A}, \mathcal{T}')$, where $\mathcal{C}' := \mathcal{C} \cup \{C\}$ with $C \notin \mathcal{C}$ and $\mathcal{T}' := \mathcal{T} \cup \{C \doteq E\}$. Since $|\mathcal{S}'|$ is linear in $|\mathcal{S}| + |E|$, it is indeed sufficient to consider the case where E is a class name C.

Let $\mathcal{S}_{rel}$ be the relaxation of $\mathcal{S}$. By Propositions 5.1.3 and 5.1.14, C is consistent in $\mathcal{S}$ if and only if it is consistent in $\mathcal{S}_{rel}$. By Theorem B.2.3, C is consistent in $\mathcal{S}_{rel}$ if and only if the CPDL-formula $f := \delta^+(\mathcal{S}_{rel}) + \delta(C)$ is satisfiable. $|f|$ is linear in $|\mathcal{S}_{rel}|$ and by Lemma 5.1.2 linear in $|\mathcal{S}|$. The claim then follows from the fact that satisfiability in CPDL can be decided in worst case deterministic exponential time [127]. □

Corollary 5.1.16 *Unrestricted class consistency in primitive $\mathcal{LUNI}$-schemata is **EXPTIME**-complete.*

Proof. The claim follows from Corollary 4.3.5 and Theorem 5.1.15. □

The method described in this section can be used also to decide unrestricted class subsumption $\mathcal{S} \models_f E_1 \preceq E_2$ in relevant cases. In particular, the condition that has to be ensured for our method to work is the existence of a two-way deterministic tree model for the satisfiable CPDL-formula corresponding to the subsumption problem. By Theorem B.2.3, $\mathcal{S} \models_f E_1 \preceq E_2$ if and only if $f := \delta^+(\mathcal{S}) \wedge \delta(E_1) \wedge \neg\delta(E_2)$ is unsatisfiable. If we allow for E_2 an arbitrary class expression, due to presence of the negated sub-formula $\neg\delta(E_2)$, in general we obtain for f a CPDL-formula that is not primitive, since it may contain an eventuality $\langle r\rangle f'$ with $f' \neq \top$. Therefore, in general, f does not admit a two-way deterministic tree model even if it satisfiable. Obviously, if E_2 does not contain any universal quantification, the negation normal form[4] of $\neg\delta(E_2)$ is primitive, and Lemma 5.1.8 can be applied.

We can however generalize Lemma 5.1.8 also to the case where f has exactly one eventuality $\langle r\rangle f'$ with $f' \neq \bot$. Intuitively this is due to the fact that we can guide the construction of the two-way deterministic tree structure $\mathcal{T}$ from an arbitrary model of f by choosing at each step the "correct" successor node to include in the tree structure. In such a way we can ensure that $\mathcal{T}$ is in fact a model of f. The other results easily generalize to this case. Since these results are subsumed by the ones in Section 5.2, we omit the details of the construction and the generalizations of the lemmata, and state the final result without proofs.

Theorem 5.1.17 *Unrestricted class subsumption $\mathcal{S} \models_u E_1 \preceq E_2$ in primitive $\mathcal{LUNI}$-schemata can be decided in worst case deterministic exponential time in $|S| + |E_1| + |E_2|$, assuming E_2 contains at most one constructor for universal quantification.* □

5.2 Unrestricted Model Reasoning on Free $\mathcal{LT}^-$-Schemata

We have seen in Section 4.4 that several of the constructors present in $\mathcal{LT}$, when used in an unrestricted way, give rise to undecidability of the class consistency problem. In this section we show that in $\mathcal{LT}^-$ instead, the careful choice of restrictions on the use of constructors is sufficient to avoid all problems encountered in $\mathcal{LT}$. The main result of this section is in fact an algorithm to decide class consistency in free $\mathcal{LT}^-$-schemata that works in deterministic exponential time.

Table 5.2 recalls the constructors and assertions that are allowed in free $\mathcal{LT}^-$-schemata, and in the rest of the section we assume that the schemata we deal with are of this type.

The reasoning procedure we devise for class consistency in $\mathcal{LT}^-$ can be divided in three main steps. In the first step we reduce reasoning on free $\mathcal{LT}^-$-schemata to reasoning on free $\mathcal{LCQIL}\Delta$-schemata, by "reifying" basic links, tuples and sets, and therefore flattening all objects to individuals. Successively we argue that qualified number restrictions can in fact be replaced by functional restrictions only, thus reducing the problem to a class consistency problem in $\mathcal{LCFIL}\Delta$. Finally we show that this problem is decidable and can be solved in deterministic exponential time by exploiting automata theoretic techniques developed in the context of PDLs.

5.2.1 Reduction from $\mathcal{LT}^-$ to $\mathcal{LCQIL}\Delta$

We define now the reified counterpart of a schema in which basic links tuples and sets are replaced by $\mathcal{LCQIL}\Delta$-class expressions. The reification is done by defining suitable mappings from $\mathcal{LT}^-$-class and link expressions to $\mathcal{LCQIL}\Delta$-class and link expressions and by using these mappings to construct the assertions of the $\mathcal{LCQIL}\Delta$-schema. The mappings defined here extend the ones used in [58] in order to handle the repeat constructor.

Construction 5.2.1 Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be an $\mathcal{LT}^-$-schema. The *reified counterpart* $\wp(\mathcal{S}) := (\wp(\mathcal{C}), \wp(\mathcal{A}), \wp(\mathcal{T}))$ of $\mathcal{S}$ is constructed as follows. Let

$$\begin{aligned}
\mathcal{C}_{\mathcal{A}} &:= \{\top_A \mid A \in \mathcal{A}\} \cup \{\top_\ni\} \\
\mathcal{C}_{\{\}} &:= \{\top_{\{\}}\} \\
\mathcal{C}_{[]} &:= \{\top_{[]}\} \cup \{\top_{[A_1,\ldots,A_k]} \mid [A_1,\ldots,A_k] \text{ occurs in } \mathcal{T}\} \cup \\
&\quad\ \{\top_{[A_1,\ldots,A_k]} \mid \langle E \mid A_1,\ldots,A_k\rangle \text{ occurs in } \mathcal{T} \text{ for some } E\} \\
\mathcal{C}_{\langle\rangle} &:= \{\top_{\langle E|A_1,\ldots,A_k\rangle} \mid \langle E \mid A_1,\ldots,A_k\rangle \text{ occurs in } \mathcal{T}\}.
\end{aligned}$$

[4] For PDL the negation normal form of a formula is defined exactly as for the corresponding $\mathcal{L}$-language.

Constructor Name	Syntax	Semantics
class name	C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
set	$\{E\}$	$\{\{\!\{o_1, \ldots, o_h\}\!\} \in \mathcal{O}^{\mathcal{I}} \mid o_1, \ldots, o_h \in E^{\mathcal{I}}\}$
tuple	$[A_1, \ldots, A_k]$	$\{[A_1\!:o_1, \ldots, A_k\!:o_k] \in \mathcal{O}^{\mathcal{I}} \mid o_1, \ldots, o_k \in \mathcal{O}^{\mathcal{I}}\}$
tuple with key	$\langle E \mid A_1, \ldots, A_k\rangle$	$\subseteq [A_1, \ldots, A_k]^{\mathcal{I}} \cap E^{\mathcal{I}}$ and key condition
conjunction	$E_1 \sqcap E_2$	$E_1^{\mathcal{I}} \cap E_2^{\mathcal{I}}$
general negation	$\neg E$	$\Delta^{\mathcal{I}} \setminus E^{\mathcal{I}}$
universal quantif.	$\forall A.E$	$\{o \mid \forall o' \colon (o, o') \in L^{\mathcal{I}} \rightarrow o' \in E^{\mathcal{I}}\}$
qualified number restrictions	$\exists^{\leq n} B.E$	$\{o \mid \sharp\{o' \mid (o, o') \in B^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}}\} \leq n\}$
	$\exists^{\leq n} B^-.E$	$\{o \mid \sharp\{o' \mid (o', o) \in B^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}}\} \leq n\}$
role value map	$(B_1 \subseteq B_2)$	$\{o \mid \{o' \mid (o, o') \in B_1^{\mathcal{I}}\} \subseteq \{o' \mid (o, o') \in B_2^{\mathcal{I}}\}\}$
	$(B_1^- \subseteq B_2^-)$	$\{o \mid \{o' \mid (o', o) \in B_1^{\mathcal{I}}\} \subseteq \{o' \mid (o', o) \in B_2^{\mathcal{I}}\}\}$
repeat	$\Delta(L)$	$\{o_0 \mid \exists o_1, o_2, \ldots : (o_i, o_{i+1}) \in L^{\mathcal{I}}, i \geq 0\}$
attribute name	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
member	$\ni$	$\{(\{\!\{\ldots, o, \ldots\}\!\}, o) \in \mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}\}$
union	$L_1 \cup L_2$	$L_1^{\mathcal{I}} \cup L_2^{\mathcal{I}}$
difference	$B_1 \setminus B_2$	$B_1^{\mathcal{I}} \setminus B_2^{\mathcal{I}}$
concatenation	$L_1 \circ L_2$	$L_1^{\mathcal{I}} \circ L_2^{\mathcal{I}}$
inverse	L^-	$\{(o, o') \mid (o', o) \in L^{\mathcal{I}}\}$
transitive closure	L^*	$(L^{\mathcal{I}})^*$
identity	$id(E)$	$\{(o, o) \mid o \in E^{\mathcal{I}}\}$
free assertion	$E_1 \dot{\preceq} E_2$	$E_1^{\mathcal{I}} \subseteq E_2^{\mathcal{I}}$

Table 5.2: Free $\mathcal{LT}^-$-schemata

The sets of classes and attributes of $\wp(\mathcal{S})$ are defined as follows:

$$\begin{aligned} \wp(\mathcal{C}) &:= \mathcal{C} \cup \mathcal{C}_{\mathcal{A}} \cup \mathcal{C}_{\{\}} \cup \mathcal{C}_{[]} \cup \mathcal{C}_{\langle\rangle} \cup \{\top_{\mathcal{C}}, \top_{\mathcal{L}}\} \\ \wp(\mathcal{A}) &:= \{V_1, V_2\}. \end{aligned}$$

In order to construct $\wp(\mathcal{T})$ we need to define a mapping γ from $\mathcal{LT}^-$-class expressions to $\mathcal{LCQIL}\Delta$-class expressions. This is done by defining inductively three mappings.

- The mapping γ'' from $\mathcal{LT}^-$-basic links to $\mathcal{LCQIL}\Delta$-class expressions is defined as follows:

$$\begin{aligned} \gamma''(A) &:= \top_A \\ \gamma''(\ni) &:= \top_{\ni} \\ \gamma''(B_1 \cup B_2) &:= \gamma''(B_1) \sqcup \gamma''(B_2) \\ \gamma''(B_1 \setminus B_2) &:= \gamma''(B_1) \sqcap \neg\gamma''(B_2). \end{aligned}$$

- The mapping γ' from $\mathcal{LT}^-$-links to $\mathcal{LCQIL}\Delta$-links is defined as follows:

$$\begin{aligned} \gamma'(B) &:= V_1^- \circ id(\gamma''(B)) \circ V_2 \\ \gamma'(L_1 \circ L_2) &:= \gamma'(L_1) \circ \gamma'(L_2) \\ \gamma'(L_1 \cup L_2) &:= \gamma'(L_1) \cup \gamma'(L_2) \\ \gamma'(L^*) &:= (\gamma'(L))^* \\ \gamma'(L^-) &:= (\gamma'(L))^- \\ \gamma'(id(E)) &:= id(\gamma(E)). \end{aligned}$$

- Finally, the mapping γ from $\mathcal{LT}^-$-class expressions to $\mathcal{LCQIL}\Delta$-class expression is defined as follows:

$$\begin{aligned} \gamma(C) &:= C \\ \gamma(\{E\}) &:= \top_{\{\}} \sqcap \forall(V_1^- \circ id(\top_{\ni}) \circ V_2).\gamma(E) \end{aligned}$$

$$
\begin{array}{rcl}
\gamma([A_1,\ldots,A_k]) & := & \top_{[A_1,\ldots,A_k]} \\
\gamma(\langle E \mid A_1,\ldots,A_k\rangle) & := & \top_{\langle E|A_1,\ldots,A_k\rangle} \\
\gamma(E_1 \sqcap E_2) & := & \gamma(E_1) \sqcap \gamma(E_2) \\
\gamma(\neg E) & := & \neg\gamma(E) \\
\gamma(\forall L.E) & := & \forall\gamma'(L).\gamma(E) \\
\gamma(\exists^{\leq n} B.E) & := & \exists^{\leq n} V_1^-.(\gamma''(B) \sqcap \forall V_2.\gamma(E)) \\
\gamma(\exists^{\leq n} B^-.E) & := & \exists^{\leq n} V_2^-.(\gamma''(B) \sqcap \forall V_1.\gamma(E)) \\
\gamma(\Delta(L)) & := & \Delta(\gamma'(L)) \\
\gamma((B_1 \subseteq B_2)) & := & \forall V_1.(\neg\gamma''(B_1) \sqcup \gamma''(B_2)) \\
\gamma((B_1^- \subseteq B_2^-)) & := & \forall V_2.(\neg\gamma''(B_1) \sqcup \gamma''(B_2)).
\end{array}
$$

We introduce some abbreviations:

$$
\begin{array}{rcl}
memb & := & V_1^- \circ id(\top_\ni) \circ V_2 \\
P_A & := & V_1^- \circ id(\top_A) \circ V_2 \\
\exists^{\leq n} P_A.E & := & \exists^{\leq n} V_1^-.(\top_A \sqcap \forall V_2.E) \\
\exists^{=n} P_A.E & := & \exists^{\leq n} V_1^-.(\top_A \sqcap \forall V_2.E) \sqcap \exists P_A.E \\
\exists^{\leq n} P_A^-.E & := & \exists^{\leq n} V_2^-.(\top_A \sqcap \forall V_1.E) \\
\exists^{=n} P_A^-.E & := & \exists^{\leq n} V_2^-.(\top_A \sqcap \forall V_1.E) \sqcap \exists P_A^-.E \\
\top_{[|A_i|]} & := & \top_{[A_i]} \sqcap \forall(P_{A_1} \cup \cdots \cup P_{A_{i-1}} \cup P_{A_{i+1}} \cup \cdots \cup P_{A_h}).\bot, \\
 & & \quad \text{if } \mathcal{A} := \{A_1,\ldots,A_h\}.
\end{array}
$$

The set $\wp(\mathcal{T})$ of assertions of $\wp(\mathcal{S})$ is defined as $\wp(\mathcal{T}) := \wp_1(\mathcal{T}) \cup \wp_2(\mathcal{T}) \cup \wp_3(\mathcal{T})$, where

- $\wp_1(\mathcal{T}) := \{\gamma(E_1) \dot{\preceq} \gamma(E_2) \mid E_1 \dot{\preceq} E_2 \in \mathcal{T}\}$.
- $\wp_2(\mathcal{T})$ is the set constituted by the assertions:

$$
\begin{array}{rcll}
\top_{\{\}} & \doteq & \exists memb.\top & (5.1) \\
\top_{\{\}} \sqcap \exists^{\leq 1} memb.\top & \dot{\preceq} & \forall memb.(\exists^{\leq 1} memb^-.(\exists^{\leq 1} memb.\top)). & (5.2) \\
\top_{[|A|]} & \dot{\preceq} & \exists^{=1} P_A.\top \sqcap \forall P_A.(\exists^{\leq 1} P_A^-.[|A|]), & (5.3) \\
 & & \quad \text{for each } \top_{[A]} \text{ in } \mathcal{C}_{[]} & \\
\top_{[A_1,\ldots,A_k]} & \doteq & \top_{[]} \sqcap \exists^{=1} P_{A_1}.\top \sqcap \cdots \sqcap \exists^{=1} P_{A_k}.\top, & (5.4) \\
 & & \quad \text{for each } \top_{[A_1,\ldots,A_k]}, \text{ with } k \geq 1, \text{ in } \mathcal{C}_{[]} & \\
\top_{\langle E|A\rangle} & \dot{\preceq} & \gamma(E) \sqcap \top_{[A]} \sqcap \forall P_A.(\exists^{\leq 1} P_A^-.\gamma(E)), & (5.5) \\
 & & \quad \text{for each } \top_{\langle E|A\rangle} \text{ in } \mathcal{C}_{\langle\rangle} & \\
\top_{\langle E|A_1,\ldots,A_k\rangle} & \dot{\preceq} & \gamma(E) \sqcap \top_{[A_1,\ldots,A_k]}, & (5.6) \\
 & & \quad \text{for each } \top_{\langle E|A_1,\ldots,A_k\rangle}, \text{ with } k \geq 2, \text{ in } \mathcal{C}_{\langle\rangle}. &
\end{array}
$$

- $\wp_3(\mathcal{T})$ is the set constituted by the assertions:

$$
\begin{array}{rcll}
\top & \dot{\preceq} & \top_{\mathcal{C}} \sqcup \top_{\mathcal{L}} & (5.7) \\
\top_{\mathcal{C}} & \dot{\preceq} & \forall V_1.\bot \sqcap \forall V_2.\bot & (5.8) \\
\top_{\mathcal{L}} & \dot{\preceq} & \exists V_1.\top_{\mathcal{C}} \sqcap \exists^{\leq 1} V_1.\top \sqcap \exists V_2.\top_{\mathcal{C}} \sqcap \exists^{\leq 1} V_2.\top & (5.9) \\
C & \dot{\preceq} & \top_{\mathcal{C}}, \quad \text{for each } C \in \mathcal{C} \cup \mathcal{C}_{\{\}} \cup \mathcal{C}_{[]} & (5.10) \\
\top_A & \dot{\preceq} & \top_{\mathcal{L}}, \quad \text{for each } \top_A \in \mathcal{C}_{\mathcal{A}}. & (5.11)
\end{array}
$$

■

Lemma 5.2.2 *The reified counterpart $\wp(\mathcal{S})$ of $\mathcal{S}$ can be constructed in time which is polynomial in $|\mathcal{S}|$.*

Proof. Straightforward. □

Observe that $|\wp(\mathcal{S})|$ is polynomially related to $|\mathcal{S}|$, and that $\wp(\mathcal{S})$ is interpreted over usual interpretations $\mathcal{I} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a set of unstructured objects. The idea behind the construction of $\wp(\mathcal{S})$ is that for each model $\mathcal{I}$ of $\mathcal{S}$ there should be a corresponding model $\wp(\mathcal{I})$ of $\wp(\mathcal{S})$, in which all basic links, sets, and tuples of $\mathcal{I}$ are reified, i.e. correspond to unstructured objects in $\Delta^{\wp(\mathcal{I})}$. In particular, all attribute names in $\mathcal{A}$ are reified. Each pair $(o_1, o_2) \in A^{\mathcal{I}}$ is represented in $\wp(\mathcal{I})$ by an object $o \in \top_A^{\wp(\mathcal{I})}$ connected through the attributes V_1 and V_2 to the objects representing o_1 and o_2 respectively. For sets and tuples we have in fact embedded a "double reification" in the construction of $\wp(\mathcal{S})$. For example, a set $\{\!|o_1, \ldots, o_k|\!\} \in \mathcal{O}^{\mathcal{I}}$ is represented in $\Delta^{\wp(\mathcal{I})}$ by an object $o' \in \top_{\{\}}^{\wp(\mathcal{I})}$ connected through link $\ni$ to the objects $o'_1, \ldots, o'_k$ representing $o_1, \ldots, o_k$ respectively. The link $\ni$ is then itself reified, and each pair of objects $(o', o_i) \in \ni^{\wp(\mathcal{I})}$ is replaced by a new object $o''_i \in \top_{\ni}^{\wp(\mathcal{I})}$ connected through attributes V_1 and V_2 to the objects representing o' and o_i respectively. A similar observation holds for objects representing tuples and tuples with key. Therefore it is sufficient to use only the two attributes V_1 and V_2 in $\wp(\mathcal{A})$.

It would be nice to establish now a one to one mapping from models $\mathcal{I}$ of $\mathcal{S}$ to models $\wp(\mathcal{I})$ of $\wp(\mathcal{S})$. The following observation shows, however, that this is not possible in general. In a model of an $\mathcal{LT}^-$-schema $\mathcal{S}$ each tuple and set is unique, while in a model of $\wp(\mathcal{S})$ there may be more than one object representing the same set or tuple. For example, there may be two objects $o, o' \in \top_{\{\}}^{\wp(\mathcal{I})}$, connected through *memb* to exactly the same objects $o'_1, \ldots, o'_k \in \Delta^{\wp(\mathcal{I})}$. In this case o and o' both represent the same set $\{\!|o_1, \ldots, o_k|\!\} \in \mathcal{O}^{\mathcal{I}}$, being $o'_1, \ldots, o'_k \in \Delta^{\wp(\mathcal{I})}$ the objects representing $o_1, \ldots, o_k \in \mathcal{O}^{\mathcal{I}}$, respectively. The same problem may occur with a reified attribute where two objects in $\top_{\mathcal{L}}^{\wp(\mathcal{I})}$ may represent the same pair of objects in $\mathcal{O}^{\mathcal{I}}$. Therefore, any model of $\wp(\mathcal{S})$ in which such a situation occurs does not correspond directly to a model of $\mathcal{S}$.

We characterize now those models of $\wp(\mathcal{S})$ that behave "nicely" and show that there is indeed a one to one correspondence between these models of $\wp(\mathcal{S})$ and the models of $\mathcal{S}$. The situation is similar to the one encountered in Section 3.3, where we characterized (in Definition 3.3.6) those models that correctly represent arbitrary n-ary relations.

Definition 5.2.3 A model $\mathcal{I}$ of $\wp(\mathcal{S})$ is an *aggregate-descriptive model*, if the following conditions hold:

- For every $o, o' \in \top_{\mathcal{L}}^{\mathcal{I}}$ and $o_1, o_2 \in \Delta^{\mathcal{I}}$,

$$((o, o_1) \in V_1^{\mathcal{I}} \wedge (o, o_2) \in V_2^{\mathcal{I}} \wedge (o', o_1) \in V_1^{\mathcal{I}} \wedge (o', o_2) \in V_2^{\mathcal{I}}) \rightarrow o = o'. \tag{5.12}$$

- For every $o, o' \in \top_{\{\}}^{\mathcal{I}}$,

$$o \neq o' \rightarrow (\exists o'' \in \Delta^{\mathcal{I}} : (o, o'') \in memb^{\mathcal{I}} \wedge (o', o'') \notin memb^{\mathcal{I}}). \tag{5.13}$$

- For every $o, o' \in \top_{[]}^{\mathcal{I}}$,

$$(\forall o'' \in \Delta^{\mathcal{I}} : \bigwedge_{1 \leq i \leq h} ((o, o'') \in P_{A_i}^{\mathcal{I}} \leftrightarrow (o', o'') \in P_{A_i}^{\mathcal{I}})) \rightarrow o = o', \tag{5.14}$$

  where $\{A_1, \ldots, A_h\} = \mathcal{A}$.

- For every $\top_{\langle E|A_1,\ldots,A_k\rangle} \in \mathcal{C}_{\langle\rangle}$, for every $o, o' \in \top_{\langle E|A_1,\ldots,A_k\rangle}^{\mathcal{I}}$,

$$(\bigwedge_{1 \leq i \leq k} ((o, o_i) \in P_{A_i}^{\mathcal{I}}) \wedge \bigwedge_{1 \leq i \leq k} ((o', o_i) \in P_{A_i}^{\mathcal{I}})) \rightarrow o = o'. \tag{5.15}$$

■

Lemma 5.2.4 *Let $\mathcal{S}$ be an $\mathcal{LT}^-$-schema, and $\wp(\mathcal{S})$ be its reified counterpart. Then there is a one to one mapping between the models of $\mathcal{S}$ and the aggregate-descriptive models of $\wp(\mathcal{S})$.*

Proof (sketch). Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ and $\wp(\mathcal{S}) := (\wp(\mathcal{C}), \wp(\mathcal{A}), \wp(\mathcal{T}))$. For the first direction of the proof, let $\mathcal{I} := (\mathcal{O}^\mathcal{I}, \cdot^\mathcal{I})$ be a model of $\mathcal{S}$. We construct an interpretation $\mathcal{I}' := (\Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'})$ of $\wp(\mathcal{S})$ by defining

$$\Delta^{\mathcal{I}'} \;:=\; \mathcal{O}^\mathcal{I} \cup \{o_{(o_1,o_2)} \mid (o_1, o_2) \in A^\mathcal{I}, \text{ for some } A \in \mathcal{A} \cup \{\exists\}\}.$$

Each class name is interpreted in $\mathcal{I}'$ as in $\mathcal{I}$, while the class names $\top_{[A_1,\ldots,A_k]}$ and $\top_{\langle E|A_1,\ldots,A_k\rangle}$ are interpreted as the corresponding class expressions $[A_1, \ldots, A_k]$, and $\langle E \mid A_1, \ldots, A_k\rangle$. The only two attributes V_1 and V_2 are interpreted as follows:

$$\begin{aligned} V_1^{\mathcal{I}'} &:= \{(o_{(o_1,o_2)}, o_1) \mid o_{(o_1,o_2)}, o_1 \in \Delta^{\mathcal{I}'}\} \\ V_2^{\mathcal{I}'} &:= \{(o_{(o_1,o_2)}, o_2) \mid o_{(o_1,o_2)}, o_2 \in \Delta^{\mathcal{I}'}\}. \end{aligned}$$

Using the definitions of mappings γ, γ', and γ'' it is not difficult to see that since $\mathcal{I}$ is a model of $\mathcal{S}$, $\mathcal{I}'$ satisfies all assertions in $\wp_1(\mathcal{T})$. Assertion 5.1 is satisfied if we interpret $\top_{\{\}}$ as the set of all sets in $\mathcal{O}^\mathcal{I}$. Assertions 5.3 are satisfied since in each model of $\mathcal{S}$ all tuples, and in particular tuples that have just one component are unique. A similar observation holds for assertions 5.2 and 5.5. Assertions 5.4 and 5.6 are satisfied by the semantics of tuples and keys. Finally, assertions 5.7 to 5.11 are satisfied if we interpret $\top_\mathcal{C}$ as $\mathcal{O}^\mathcal{I}$, and $\top_\mathcal{L}$ as the set of objects in $\Delta^{\mathcal{I}'}$ representing pairs of objects in $\Delta^\mathcal{I}$.

For the other direction of the proof, let $\mathcal{I}' := (\Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'})$ be an aggregate-descriptive model of $\wp(\mathcal{S})$. A model $\mathcal{I} := (\mathcal{O}^\mathcal{I}, \cdot^\mathcal{I})$ of $\mathcal{S}$ can be obtained from $\mathcal{I}'$ by defining $\mathcal{O}^\mathcal{I} := \top_\mathcal{C}^{\mathcal{I}'}$. Each class name is interpreted in $\mathcal{I}$ as in $\mathcal{I}'$. All objects that are instances of $\top_{[]}$ in $\mathcal{I}'$, have in $\mathcal{I}$ the structure of a tuple, and all objects that are instances of $\top_{\{\}}$ in $\mathcal{I}'$, have in $\mathcal{I}$ the structure of a set. Every attribute $A \in \mathcal{A}$ is interpreted as the set of pairs of objects $(o_1, o_2) \in \Delta^{\mathcal{I}'}$ such that for some $o \in \top_A^{\mathcal{I}'}$, $(o, o_1) \in V_1^{\mathcal{I}'}$ and $(o, o_2) \in V_2^{\mathcal{I}'}$. Notice that assertions 5.7 to 5.11 imply that $o_1, o_2 \in \mathcal{O}^\mathcal{I}$, while $o \notin \mathcal{O}^\mathcal{I}$. Also, since $\mathcal{I}'$ is an aggregate descriptive model, the cardinality of $A^\mathcal{I}$ is equal to the cardinality of $\top_A^{\mathcal{I}'}$, which implies that also all number restrictions are interpreted in the correct way. A similar observation holds for tuples and sets, where additionally properties 5.13, 5.14 and 5.15 guarantee that sets and tuples are unique, and that the key conditions are satisfied in $\mathcal{I}$. Using also the assertions in $\wp_1(\mathcal{T})$ it is easy to see, that $\mathcal{I}$ is a model of $\mathcal{S}$. □

Lemma 5.2.4 implies that for a class name $C \in \mathcal{C}$, $\mathcal{S} \not\models_u C \equiv \bot$ is equivalent to verifying if C is consistent in an aggregate descriptive model of $\wp(\mathcal{S})$. However, this does not directly establish decidability of class consistency in $\mathcal{LT}^-$, since it is not known how to reason with the reified counterpart $\wp(\mathcal{S})$ of an $\mathcal{LT}^-$-schema $\mathcal{S}$ when interpreted over aggregate-descriptive models.

Following [58] we show now that this problem can be overcome in a very simple way, namely by just forgetting about aggregate descriptive models. This is a consequence of Lemma 5.2.5 below which shows that it is always possible to interpret a schema over an aggregate-descriptive model when verifying class consistency. The proof of Lemma 5.2.5 is based on the *disjoint union model property* which can be stated as follows: Let $\mathcal{S}$ be a $\mathcal{LT}^-$-schema and $\mathcal{I} := (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ and $\mathcal{I}' := (\Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'})$ be two models of $\mathcal{S}$. Then the interpretation $\mathcal{I} \uplus \mathcal{I}' := (\Delta^\mathcal{I} \uplus \Delta^{\mathcal{I}'}, \cdot^\mathcal{I} \uplus \cdot^{\mathcal{I}'})$ obtained as the disjoint union of $\mathcal{I}$ and $\mathcal{I}'$, is also a model of $\mathcal{S}$. The disjoint union model property is in fact typical of the vast majority of Modal Logics and of all logics whose semantics is based on Kripke structures, such as Description Logics.

Lemma 5.2.5 *Let $\mathcal{S}$ be an $\mathcal{LT}^-$-schema and $\wp(\mathcal{S})$ its reified counterpart. If $\wp(\mathcal{S})$ is consistent, then it has an aggregate-descriptive model.*

Proof (sketch). Let $\mathcal{I}$ be a model of $\wp(\mathcal{S})$. We can build an aggregate-descriptive model $\mathcal{I}'$ as follows:

We start by transforming $\mathcal{I}$ into a model $\mathcal{I}^r$ satisfying condition 5.12. Given an object $a \in \top_\mathcal{L}^\mathcal{I}$, we denote by $V_i(a)$, $i \in \{1, 2\}$, the object o such that $(a, o) \in V_i^\mathcal{I}$ (recall that due to assertion 5.9, $V_i^\mathcal{I}$ is functional). For each $o_1, o_2 \in \Delta^\mathcal{I}$ we define $X_{(V_1:o_1,V_2:o_2)} := \{a \in \top_\mathcal{L}^\mathcal{I} \mid V_1(a) = o_1 \wedge V_2(a) = o_2\}$. Some of such sets will be empty, some will be a singleton, some will be not a singleton. We call *conflict* the existence of a non-singleton set $X_{(V_1:o_1,V_2:o_2)}$. From each non-singleton $X_{(V_1:o_1,V_2:o_2)}$ we randomly choose one object o, and we say that the others *induce a conflict* on $(V_1\colon o_1, V_2\colon o_2)$. We call *Conf* the set of all objects inducing a conflict for some $(V_1\colon o_1, V_2\colon o_2)$. Note that *Conf* may be uncountable, in general.

We define an interpretation $\mathcal{I}_{2^{Conf}}$ as the disjoint union of $\sharp(2^{Conf})$ copies of $\mathcal{I}$, one copy, denoted by $\mathcal{I}_{\mathcal{Z}}$, for every set $\mathcal{Z} \in 2^{Conf}$. We denote by $o_{\mathcal{Z}}$ the copy in $\mathcal{I}_{\mathcal{Z}}$ of the object o in $\mathcal{I}$. By the disjoint union model property, $\mathcal{I}_{2^{Conf}}$ is a model of $\wp(\mathcal{S})$. Let $\mathcal{I}_{\mathcal{Z}}$ and $\mathcal{I}_{\mathcal{Z}'}$ be two copies of $\mathcal{I}$ in $\mathcal{I}_{2^{Conf}}$. We call *exchanging* $V_2(a_{\mathcal{Z}})$ *with* $V_2(a_{\mathcal{Z}'})$ the operation on $\mathcal{I}_{2^{Conf}}$ consisting of replacing in $V_2^{\mathcal{I}_{\mathcal{Z}}}$ $(a_{\mathcal{Z}}, V_2(a_{\mathcal{Z}}))$ with $(a_{\mathcal{Z}}, V_2(a_{\mathcal{Z}'}))$ and, at the same time, replacing in $V_2^{\mathcal{I}_{\mathcal{Z}'}}$ $(a_{\mathcal{Z}'}, V_2(a_{\mathcal{Z}'}))$ with $(a_{\mathcal{Z}'}, V_2(a_{\mathcal{Z}}))$. Intuitively, by exchanging $V_2(a_{\mathcal{Z}})$ with $V_2(a_{\mathcal{Z}'})$, the objects $a_{\mathcal{Z}}$ and $a_{\mathcal{Z}'}$ do not induce conflicts anymore. $\mathcal{I}^r$ is now obtained from $\mathcal{I}_{2^{Conf}}$ as follows: For each $a \in Conf$, for each $\mathcal{Z} \in 2^{Conf}$ such that $a \in \mathcal{Z}$, we exchange $V_2(a_{\mathcal{Z}})$ with $V_2(a_{\mathcal{Z}\setminus\{a\}})$. It is possible to show that all conflicts are thus eliminated while no new conflict is created. Hence, in $\mathcal{I}^r$, condition 5.12 is satisfied. We still have to show that $\mathcal{I}^r$ is a model of $\wp(\mathcal{S})$. Indeed, it is straightforward to check by induction that for every class expression E appearing in $\wp(\mathcal{S})$, for all $\mathcal{Z} \in 2^{Conf}$, $o \in E^{\mathcal{I}}$ if and only if $d_{\mathcal{Z}} \in E^{\mathcal{I}^r}$. Thus all assertions in $\wp(\mathcal{T})$ are still satisfied in $\mathcal{I}^r$.

The above transformation can be adapted in order to build from $\mathcal{I}^r$ a model $\mathcal{I}^{rs}$ fulfilling conditions 5.13 and 5.14. Observe that for singleton sets and for tuples with only one component, conditions 5.13 and 5.14 are already fulfilled in $\mathcal{I}$ due to assertions 5.2 and 5.3 in $\wp(\mathcal{T})$.

Finally, the aggregate-descriptive model $\mathcal{I}'$ is obtained from $\mathcal{I}^{rs}$ as follows: We enumerate in some way the various $\top_{\langle E|A_1,\ldots,A_k\rangle}$, with $n \geq 2$, in $\top_{[]}$ (there are a finite number, say l, of them), and we apply l times, one for each $\top_{\langle E|A_1,\ldots,A_k\rangle}$, a variant of transformation above to $\mathcal{I}^{rs}$, so that at each step h of this process, the resulting model of $\wp(\mathcal{S})$ is forced to satisfy condition 5.15 for the h-th $\top_{\langle E|A_1,\ldots,A_k\rangle}$. □

Summing up, we can state the following proposition.

Proposition 5.2.6 *Let $\mathcal{S}$ be an $\mathcal{LT}^-$-schema, C a class name in $\mathcal{S}$, and $\wp(\mathcal{S})$ the reified counterpart of $\mathcal{S}$, obtained from $\mathcal{S}$ according to Construction 5.2.1. Then C is consistent in $\mathcal{S}$ if and only if it is consistent in $\wp(\mathcal{S})$.*

Proof. "⇒" Straightforward by Lemma 5.2.4.

"⇐" Let $\mathcal{I}$ be a model of $\wp(\mathcal{S})$ with $C^{\mathcal{I}} \neq \emptyset$. By Lemma 5.2.5 there is an aggregate descriptive model $\mathcal{I}'$ of $\wp(\mathcal{S})$ with $C^{\mathcal{I}'} \neq \emptyset$. By Lemma 5.2.4 $\mathcal{I}'$ corresponds to a model $\mathcal{I}''$ of $\mathcal{S}$ and $C^{\mathcal{I}''} \neq \emptyset$. □

5.2.2 Reduction from $\mathcal{LCQIL}\Delta$ to $\mathcal{LCFIL}\Delta$

The reified counterpart $\wp(\mathcal{S})$ of an $\mathcal{LT}^-$-schema $\mathcal{S}$ may still contain qualified number restrictions, which are cumbersome to deal with, when applying the automata theoretic techniques we present in the next section. Using a technique introduced in [52] we show now how to transform an $\mathcal{LCQIL}\Delta$-schema into one containing just functional restrictions (on attributes and inverse attributes).

The transformation is done by representing the inverse attribute V_i^-, $i \in \{1, 2\}$, which is not functional (while V_i is so), by the attribute expression $F_i \circ R_i^*$, where F_i, R_i are new attributes which will be forced to be globally functional. The main point of such transformation is that now qualified number restrictions can be encoded as constraints on the chain $F_i \circ R_i^*$. Formally, we define the functional counterpart of an $\mathcal{LCQIL}\Delta$-schema as follows.

Construction 5.2.7 Let $\mathcal{S}$ be an $\mathcal{LT}^-$-schema and $\wp(\mathcal{S}) := (\wp(\mathcal{C}), \wp(\mathcal{A}), \wp(\mathcal{T}))$ be its reified counterpart. The *functional counterpart* $\rho(\mathcal{S}) := (\rho(\mathcal{C}), \rho(\mathcal{A}), \rho(\mathcal{T}))$ of $\mathcal{S}$ is constructed as follows:

- $\rho(\mathcal{C}) := \wp(C)$.
- $\rho(\mathcal{A}) := \{F_1F_2, R_1, R_2\}$.
- $\rho(\mathcal{T}) := \rho_1(\mathcal{T}) \cup \rho_2(\mathcal{T})$, where $\rho_1(\mathcal{T})$ is obtained from $\wp(\mathcal{T})$ by recursively replacing
  - every occurrence of V_i with $(F_i \circ R_i^*)^-$.
  - every occurrence of $\exists^{\leq 1} V_i.\top$ with $\top$.
  - every occurrence of $\exists^{\leq k} V_i^-.E$ with $\forall(F_i \circ R_i^* \circ (id(E) \circ R_i^+)^k).\neg E$, where L^+ stands for $L \circ L*$, and L^k stands for $\underbrace{L \circ \cdots \circ L}_{k \text{ times}}$.

$\rho_2(\mathcal{T})$ consists of the two assertions ($i \in \{1,2\}$):

$$\top \;\dot{\preceq}\; \exists^{\leq 1} F_i \sqcap \exists^{\leq 1} R_i \sqcap \exists^{\leq 1} F_i^- \sqcap \exists^{\leq 1} R_i^- \sqcap \neg(\exists F_i^-.\top \sqcap \exists R_i^-.\top).$$

■

Lemma 5.2.8 *The functional counterpart $\rho(\mathcal{S})$ of $\mathcal{S}$ can be constructed in time which is polynomial in $|\mathcal{S}|$.*

Proof. Straightforward, using also Lemma 5.2.2. □

Note that the assertions in $\rho_2(\mathcal{T})$ constrain the attributes F_i, R_i and their inverses to be globally functional, and impose that no object can be linked to other objects through both F_i^- and R_i^-. In other words, the link $(F_i \circ R_i^*)^-$ is functional.

Lemma 5.2.9 *Let $\mathcal{S}$ be an $\mathcal{LT}^-$-schema, $\wp(\mathcal{S})$ be its reified counterpart, and $\rho(\mathcal{S})$ be its functional counterpart. Then there is a one-to-one mapping between the models of $\rho(\mathcal{S})$ and the models of $\wp(\mathcal{S})$.*

Proof (sketch). Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$, $\wp(\mathcal{S}) := (\wp(\mathcal{C}), \wp(\mathcal{A}), \wp(\mathcal{T}))$, and $\rho(\mathcal{S}) := (\rho(\mathcal{C}), \rho(\mathcal{A}), \rho(\mathcal{T}))$. For the first direction of the proof, let $\mathcal{I}$ be a model of $\wp(\mathcal{S})$. $\mathcal{I}$ can be transformed into a model $\mathcal{I}'$ of $\rho(\mathcal{S})$ as follows. We define $\Delta^{\mathcal{I}'} := \Delta^{\mathcal{I}}$ and $C^{\mathcal{I}'} := C^{\mathcal{I}}$ for every class name $C \in \wp(\mathcal{C})$. Then, for each $o \in \Delta^{\mathcal{I}}$ such that there is some $o' \in \Delta^{\mathcal{I}}$ with $(o, o') \in (V_i^-)^{\mathcal{I}}$, let $o_1, \ldots, o_k \in \Delta^{\mathcal{I}}$ be all objects such that $(o, o_i) \in (V_i^-)^{\mathcal{I}}$. We put $(o, o_1) \in F_i^{\mathcal{I}'}$ and $(o_j, o_{j+1}) \in R_i^{\mathcal{I}'}$, for $j = 1, \ldots, l-1$. Note that since V_i is functional, $(F_i \circ R_i^*)^-$ is functional as well. By construction, the assertions in $\rho_2(\mathcal{T})$ are satisfied in $\mathcal{I}'$, and $(o, o') \in (V_i^-)^{\mathcal{I}}$ if and only if $(o, o') \in (F_i \circ R_i^*)^{\mathcal{I}'}$. Hence, considering that $o \in (\forall(F_i \circ R_i^* \circ (id(E) \circ R_i^+)^k).\neg D)^{\mathcal{I}'}$ expresses that there are at most k objects in $E^{\mathcal{I}}$ along the chain $F_i \circ R_i$, starting from o, it is easy to see that $o \in E^{\mathcal{I}}$ if and only if $o \in \rho(E)^{\mathcal{I}'}$, where $\rho(E)$ results from E by performing the transformation as for $\rho_1(\mathcal{T})$.

For the opposite direction, let $\mathcal{I}'$ be a model of $\rho(\mathcal{S})$. $\mathcal{I}'$ can be transformed in a model $\mathcal{I}$ of $\wp(\mathcal{S})$ by defining $\Delta^{\mathcal{I}} =: \Delta^{\mathcal{I}'}$, $C^{\mathcal{I}} := C^{\mathcal{I}'}$, for any class name C, and $V_i^{\mathcal{I}} := ((^-F_i \circ R_i^*))^{\mathcal{I}'}$. Note that by the assertions in $\rho_2(\mathcal{T})$, $((^-F_i \circ R_i^*))^{\mathcal{I}'}$ is a partial function. Again, considering that $o \in (\forall(F_i \circ R_i^* \circ (id(D) \circ R_i^+)^k).\neg D)^{\mathcal{I}'}$ expresses that there are at most k objects along the chain $F_i \circ R_i^*$, starting from o, which are in $D^{\mathcal{I}}$, it is easy to verify by induction that $d \in C^{\mathcal{I}}$ if and only if $d \in \rho(C)^{\mathcal{I}'}$. □

This result combined with those of Section 5.2.1 allows us to reduce reasoning in $\mathcal{LT}^-$ to reasoning in $\mathcal{LCFIL}\Delta$.

Proposition 5.2.10 *Let $\mathcal{S}$ be an $\mathcal{LT}^-$-schema, C a class name in $\mathcal{S}$, and $\rho(\mathcal{S})$ the functional counterpart of $\mathcal{S}$, obtained from $\mathcal{S}$ according to Construction 5.2.7. Then C is consistent in $\mathcal{S}$ if and only if it is consistent in $\rho(\mathcal{S})$.*

Proof. Straightforward by Proposition 5.2.6 and Lemma 5.2.9. □

5.2.3 Decidability of $\mathcal{LCFIL}\Delta$

In this section we show that verifying class consistency in free $\mathcal{LCFIL}\Delta$-schemata is decidable in deterministic exponential time in the size of the schema. To this end we exploit the correspondence defined in Section B.2 and extended in Section B.3.1 to deal with functional restrictions and the repeat constructor, and show that satisfiability in the PDL RFCPDL corresponding to $\mathcal{LCFIL}\Delta$ is decidable in deterministic exponential time.

Our decision procedure is based on automata theoretic techniques, and it is convenient to resort to the variant RFCAPDL of RFCPDL obtained by expressing programs inside eventualities by sequential automata (instead of regular expressions) and by expressing the repeat formulae by means of Büchi automata, as shown in Section B.3.2. Since the mapping ϕ from a RFCPDL-formula f to an equivalent RFCAPDL-formula $\phi(f)$ is linear, proving decidability of satisfiability in RFCAPDL in deterministic exponential time is sufficient to establish the desired result. Our proof closely follows the one given in [161] for RCADPDL and extends it to deal with local functionality on both direct and converse atomic programs. Some basic notions about automata on infinite objects have been included in Appendix A.

We reduce the problem of satisfiability of a RFCAPDL-formula to a problem of nonemptiness of the language accepted by a finite automaton on infinite trees. This is a standard technique used for proving decidability and upper bound results for PDLs [151, 161, 165], program logics [164, 163, 152] and various other modal and temporal logics [74, 85, 72, 147, 166], and which can also be profitably exploited for the optimization of database logic programs [162].

The fundamental property needed to apply these techniques is the *tree model property*, i.e. every satisfiable formula admits a tree model, and which holds also for RFCAPDL.

Proposition 5.2.11 *Every satisfiable* RFCAPDL*-formula f has an n-ary tree model $\mathcal{T}$, with n polynomial in $|f|$, and such that $\mathcal{T}, \epsilon \models f$.*

Proof. The proof can be done by standard techniques, following the line of a similar proof in [165]. □

Rather than delving into technical details, which are indeed quite intricate, in the rest of the section we would like to give an intuition on the ideas underlying the reduction from satisfiability in PDLs to nonemptiness problems for automata on infinite trees. The considerations that are specific for RFCAPDL establish then the desired reduction.

The reduction is based on the tree model property of these logics and the close correspondence that can be established between particular infinite trees that can be accepted by an automaton (Hintikka trees) and models of a formula. In fact, the automaton corresponding to a formula does not directly accept the tree models of the formula but so called Hintikka trees which are in a natural correspondence with them. A *Hintikka tree* for a formula f is an n-ary tree over an alphabet Σ that consists of all subsets of $CL^+(f) \cup X$. Here, $CL^+(f)$ denotes the "extended" closure of the formula that extends the Fischer-Ladner closure as defined in Definition B.4.1, and whose exact definition depends on the logic we are considering. X is a set of symbols that enable the automaton that accepts the Hintikka trees to verify additional conditions that need to be satisfied for establishing the correspondence with the tree models of the formula. The arity n of the Hintikka tree is at most $\sharp CL^+(f)$, which is itself polynomially related to $|f|$. A Hintikka tree can be obtained from an n-ary tree model (with n polynomial in $|f|$) by labeling each node with the set of formulae in $CL^+(f)$ satisfied in that node (and with some of the symbols in X). Note that since $\sharp CL^+(f)$ (and also $\sharp X$) is polynomial in $|f|$, the size of the alphabet of the automaton is at most exponential in $|f|$.

For this technique to work it is fundamental that the logic we consider has the tree model property, which allows us to restrict our attention to tree models. Since these are in correspondence with Hintikka trees (for a suitable definition of Hintikka tree), we can solve satisfiability in PDL by reducing it to a nonemptiness problem for automata on infinite trees. The automaton that has to be tested for nonemptiness is precisely the automaton that accepts the Hintikka trees of the formula.

Such automaton needs to check that the conditions for a Hintikka tree are satisfied. These conditions correspond to the conditions imposed by the semantics of formulae on a model. For example, if the label of a node contains a formula $f_1 \wedge f_2$, the semantics of "$\wedge$" implies that the label must contain also both formulae f_1 and f_2. Similar observations hold for the other propositional connectives, where the type of check that the automaton needs to perform is essentially local, i.e. can be performed by analyzing just the current node. The situation is fundamentally different in the case where the label of the node contains an eventuality $\langle r \rangle f$. In order to check the conditions implied by the presence of this eventuality, the automaton must be able to analyze nodes of the tree that are an arbitrary finite number of steps away from the current node. It must verify that there is indeed one such node in which f is satisfied, and which is connected to the current node via an execution sequence of r.

For this reason the automaton corresponding to a formula can usually be thought of as divided in two components [165, 85, 166]:

- A *local automaton*, which checks the local conditions.
- An *eventuality automaton*, which is responsible for verifying that all eventualities in a node are satisfied.

The definition of the local automaton is usually straightforward, while the eventuality automaton needs some more attention.

Concerning the logic we are interested in, namely RFCAPDL, we remark that local functionality on (direct and converse) atomic programs is a local condition, since it involves only the current node and its predecessor and successors. Therefore it can be checked by embedding it in the local automaton. Repeat formulae, on

the other hand, are of a different nature and cannot be handled completely neither by the local nor by the eventuality automaton. This is due to the fact that they require to check for the (non)existence of an infinite sequence of nodes satisfying certain conditions.

It turns out that the presence of the converse constructor on programs is problematic when trying to apply the method describe here. In a tree model of a PDL without converse, all eventualities are accomplished by going "downwards" in the tree. This means that if a formula $\langle r \rangle f$ is satisfied in a state x, then the sequence of states that lead to the state where f is satisfied have all the form $xx_{i_1}x_{i_2}\cdots$, i.e. they all appear below x in the tree. This fits nicely with the way tree automata operate, since the automaton needs only to go downwards in the tree to check if an eventuality is satisfied. If converse programs are present in the formula, however, going down in the tree may not be sufficient, since there may be eventualities that are accomplished in a node by moving upwards. To solve this problem, Streett proposes in [151] to use "two-way" automata to deal with converse programs and shows then how to reduce the emptiness problem for his two-way automata to the emptiness problem for classic one-way automata. Unfortunately the translation involves a triple exponential blow-up in the size of the automaton and therefore cannot lead to the single exponential upper bound we are interested in[5].

In order to deal more efficiently with converse programs and repeat, Vardi and Wolper propose in [164, 161, 165] to introduce *cycle*-formulae that deal with cycling computations. For a program r, a formula $cycle(r)$ is satisfied in a state u if there is an execution sequence of r that leads from u through a sequence of states back to u itself. Formulae of this type are not added to the logic but are just used as additional labels of the nodes of the Hintikka tree accepted by a one-way automaton. These additional labels enable the automaton to check all eventualities by moving downwards in the tree, since they "anticipate" the conditions that have to be satisfied in a node for the fulfillment of eventualities that are encountered further down. In this sense cycle-formulae act similarly to the set of axioms used in [53] to eliminate converse when verifying satisfiability.

Additionally, in order to enable one-way automata to check cycle-formulae by moving downward, one needs to add *directed cycle-formulae*, where the direction of the path that satisfies the cycle-formula is specified. The semantics of such formulae is defined only on tree structures. Finally, in order to deal with the repeat constructor, *strengthened (directed) cycle-formulae* have to be introduced, which additionally require that the automaton representing program r goes through a designated set of states when accepting the execution sequence that realizes the formula. For the details and the semantics of the introduced cycle formulae we refer to [161, 165].

Coming back to satisfiability in RFCAPDL, we remind that this logic differs from the logic RCADPDL studied in [161] by the presence of local functionality on both direct and converse atomic programs instead of global functionality on all direct atomic programs (and no restriction of this sort for converse programs). As noticed above, local functionality is a local condition, and we can extend the definition of Hintikka tree for a RCADPDL-formula f given in [161] to include it. In a Hintikka tree T for a RCADPDL-formula f, for each node x of T, the set labeling x contains besides formulae of $CL^+(f)$, also an atomic program (either direct or converse). This atomic program is used in the construction of a model $\mathcal{I}$ of f from the Hintikka tree, and denotes the program that connects in $\mathcal{I}$ the state corresponding to x to the state corresponding to the predecessor of x. The definition of Hintikka tree of [161] is modified by substituting the condition for global functionality (condition 2) with the following:

(2') For all $x \in [n]^*$:

- (2'.1) $\sharp(T(x) \cap (Prog \cup Prog^-)) \leq 1$
- (2'.2) If $b \in (Prog \cup Prog^-)$, $\langle b \rangle^{\leq 1} \in T(x)$, y and z are two distinct successors of x, and $b^- \in T(y)$, then $b^- \notin T(z)$.
- (2'.3) If $b \in (Prog \cup Prog^-)$, $\langle b \rangle^{\leq 1} \in T(x)$, y is a successors of x, and $b \in T(x)$, then $b^- \notin T(y)$.

Condition 2'.1 ensures that when constructing from T a tree model $\mathcal{I}$, the atomic program connecting a state to its predecessor can be chosen in a unique way. Condition 2'.2 ensures that if an atomic program b is required to be functional in a state u, then u cannot have two distinct successors connected to it via b. Finally Condition 2'.3 handles the case where a node is connected to its predecessor via a functional atomic program.

[5] [151] actually shows a quadruple exponential blow-up, but the complexity can be reduced by exploiting the results in [135].

Therefore, we can repeat the same arguments used in [161] for the logic RCADPDL and reduce satisfiability in RFCAPDL to a nonemptiness problem for certain types of automata. The automaton constructed from a formula accepts exactly the Hintikka trees for the formula, and therefore has to check the conditions in the definition of Hintikka tree. The local conditions, the conditions for eventualities (and cycle formulae), and a part of the conditions for repeat formulae can be verified by a Büchi tree automaton. However, one of the conditions that are essential for repeat formulae, and which is a condition that holds along infinite paths in the tree, cannot be verified by a Büchi tree automaton. The solution is to construct a Büchi sequential automaton that checks for violations of this condition. Summing up, the automaton that accepts exactly the Hintikka trees of a formula f is a pair $H_f := (R_f, B_f)$, where R_f is a Büchi tree automaton and B_f is a Büchi sequential automaton, both over $2^{CL^+(f) \cup Prog \cup Prog^- \cup \{\emptyset\}}$. R_f has a number of states that is exponential in $|f|$, while B_f has a number of states that is quadratic in $|f|$.

H_f is a special case of *hybrid tree automaton* introduced in [163], and defined as a pair $H := (R_t, B_s)$, where R_t is a Rabin tree automaton and B_s is a Büchi sequential automaton, both over the same alphabet Σ. H accepts a tree T if T is accepted by R_t, and for every infinite path p starting at ϵ, B_s rejects $T(p)$.

We can conclude these observations by stating the following proposition without proof.

Proposition 5.2.12 *Let f be a* RFCAPDL*-formula. Then one can construct in deterministic exponential time in $|f|$ a hybrid tree automaton $H_f := (R_f, B_f)$ such that the following holds:*

- *R_f has a number of states that is exponential in $|f|$.*
- *B_f has a number of states that is polynomial in $|f|$.*
- *f is satisfiable if and only if the set of trees accepted by H_f is nonempty.* □

In [163] it is shown that the nonemptiness problem for a hybrid tree automaton $H := (R_t, B_s)$ can be solved in nondeterministic time that is polynomial in the number of states of R_t and exponential in the number of states of B_s. This would give a nondeterministic exponential upper bound for satisfiability in RFCAPDL.

However, exploiting the methods in [73], which build on the fundamental results in [135], and in [72], we can obtain a deterministic exponential upper bound.

Proposition 5.2.13 (Emerson and Jutla [73]) *The nonemptiness problem for a hybrid tree automaton $H := (R_t, B_s)$ can be solved in deterministic time that is polynomial in the number of states of R_t and exponential in the number of states of B_s.*

Proof. The nonemptiness problem for $H := (R_t, B_s)$ can be solved as follows:

1. The (possibly nondeterministic) sequential Büchi automaton B_s is determinized and complemented using the construction in [73]. This yields a deterministic sequential Rabin automaton D_s with number of states exponential in the number of states of B_s, but with polynomial number of pairs.
2. D_s, which accepts all infinite strings rejected by B_s, being deterministic can easily be transformed into a Rabin tree automaton D_t which accepts the infinite trees T in which all infinite paths starting at ϵ are accepted by D_s. The number of states (pairs) of D_t is linear in the number of states (pairs) of D_s.
3. R_t and D_t can now be combined into a single Rabin tree automaton H_t that accepts the infinite trees accepted by both R_t and D_t, i.e. the infinite trees accepted by R_t and for which all infinite path starting at ϵ are rejected by B_t. Therefore H_t accepts the same set of trees as H. H_t has a number of states which is linear in the number of states of R_t and exponential in the number of states of B_s. The number of pairs of H_t is linear in the number of pairs of R_t and polynomial in the number of states of B_s.
4. By Theorem A.2.5, the nonemptiness problem for H_t, and therefore also for H, can be solved in deterministic time polynomial in the number of states of H_t and exponential in the number of pairs of H_t.

The claim follows by observing that the number of states of H_t is linear in the number of states of R_t, and the number of pairs of H_t is polynomial in the number of states of B_s. □

This allows us to establish the desired upper bound for satisfiability in RFCAPDL.

Proposition 5.2.14 *Satisfiability of a* RFCAPDL*-formula f can be decided in deterministic exponential time in $|f|$.*

Proof. By Proposition 5.2.12 we can construct from f in deterministic exponential time in $|f|$ a hybrid tree automaton $H_f := (R_f, B_f)$ such that R_f has a number of states exponential in $|f|$, B_f has a number of states polynomial in $|f|$, and f is satisfiable if and only if the set of trees accepted by H_f is nonempty. By Proposition 5.2.13 we can decide whether H_f accepts some tree in deterministic time that is polynomial in the number of states of R_f and exponential in the number of states of B_f, and therefore in deterministic exponential time in $|f|$. □

5.2.4 Upper Bounds for Unrestricted Model Reasoning

We can now integrate the results of this section and establish the upper bound for unrestricted model reasoning in $\mathcal{LT}^-$.

Theorem 5.2.15 *Unrestricted class consistency $\mathcal{S} \not\models_u E \equiv \bot$ in free $\mathcal{LT}^-$-schemata can be decided in worst case deterministic exponential time in $|\mathcal{S}| + |E|$.*

Proof. If E is not a class name, we construct the schema $\mathcal{S}' := (\mathcal{C}', \mathcal{A}, \mathcal{T}')$, where $\mathcal{C}' := \mathcal{C} \cup \{C\}$ with $C \notin \mathcal{C}$ and $\mathcal{T}' := \mathcal{T} \cup \{C \dot{\preceq} E, E \dot{\preceq} C\}$. It is easy to see that $\mathcal{S}$ and $\mathcal{S}'$ are equivalent, and that E is consistent in $\mathcal{S}$ if and only if C is consistent in $\mathcal{S}'$. Since $|\mathcal{S}'|$ is linear in $|\mathcal{S}| + |E|$, it is indeed sufficient to consider the case where E is a class name C.

By Lemma 5.2.8, we can construct the functional counterpart $\rho(\mathcal{S})$ of $\mathcal{S}$ in time polynomial in $|\mathcal{S}|$, and by Proposition 5.2.10, C is consistent in $\mathcal{S}$ if and only if it is consistent in $\rho(\mathcal{S})$. $\rho(\mathcal{S})$ is a free $\mathcal{LCFIL}\Delta$-schema and by applying the correspondence defined in Section B.2 and extended in Section B.3.1 to deal with functional restrictions and the repeat constructor, we have that C is consistent in $\rho(\mathcal{S})$ if and only if the RFCAPDL-formula $f := \delta^+(\rho(\mathcal{S})) \wedge \delta(C)$ is satisfiable. The claim follows from the fact that $|f|$ is polynomial in $|\rho(\mathcal{S})|$ and therefore also in $|\mathcal{S}|$ and by Proposition 5.2.14. □

Corollary 5.2.16 *Unrestricted class consistency in free $\mathcal{LT}^-$-schemata is **EXPTIME**-complete.*

Proof. The claim follows from Corollary 4.3.5 and Theorem 5.2.15. □

Theorem 5.2.17 *Unrestricted class subsumption $\mathcal{S} \models_u E_1 \preceq E_2$ in free $\mathcal{LT}^-$-schemata can be decided in worst case deterministic exponential time in $|S| + |E_1| + |E_2|$.*

Proof. Since $\mathcal{LT}^-$ contains general negation, for each $\mathcal{LT}^-$-class expression E, $\neg E$ is also an $\mathcal{LT}^-$-class expression. By Proposition 2.3.2, $\mathcal{S} \models_u E_1 \preceq E_2$ if and only if $\mathcal{S} \models_u E_1 \sqcap \neg E_2 \equiv \bot$, and by Theorem 5.2.15 this can be decided in worst case deterministic exponential time in $|\mathcal{S}| + |E_1| + |E_2|$. □

Chapter 6

Finite Model Reasoning

In this chapter we discuss the issues related to reasoning on $\mathcal{L}$-schemata with respect to finite models. We have seen in Chapter 2 that the presence of functional restrictions and inverse attributes in an $\mathcal{L}$-language is already sufficient to give rise to complex interactions through cyclic class specifications in a schema. Such interactions may cause a schema or a class to be consistent, but only in infinite models (see Example 2.4.2), or may cause subsumption relations to hold only if reasoning is restricted to finite models (see Example 2.4.3). The techniques based on automata on infinite trees introduced in Chapter 5 exploit the tree-model property of $\mathcal{L}$-languages and assume that even finite models are unwound to a (possibly) infinite tree. By using tree automata there seems to be no easy way to distinguish between an infinite tree-model resulting form unwinding an finite model (containing cycles) and one resulting from unwinding a proper infinite model. For this reason the known methods are not applicable in the finite case and we have to approach the problem in a completely different way. We need also a proper treatment of number restrictions which may interact with the other constructors of the language, playing a crucial role in the existence of finite models.

The technique we introduce in this chapter is based on the idea of separating the reasoning process in two distinct phases. The first phase deals with all constructors except number restrictions and existential quantifications (which in the unqualified form is just a particular case of number restriction), and builds an expanded schema in which these constructors are embedded implicitly in the classes and attributes. In the second phase the assertions involving number restrictions are used to derive from this expanded schema a system of linear inequalities. The system is defined in such a way that its solutions of a certain type (acceptable solutions) are directly related to the finite models of the original schema. In particular, from each acceptable solution one can directly deduce the cardinalities of the extensions of all classes and attributes in a possible finite model. The proposed method allows us to establish decidability of finite model reasoning in free $\mathcal{LCNI}$-schemata and provides a tight upper bound for the important case of primitive $\mathcal{LUNI}$-schemata. Moreover, under certain assumptions that are very common in databases it results in efficient reasoning procedures.

The chapter is organized as follows. Section 6.1 contains results about systems of linear inequalities that are needed in the rest of the chapter. In Section 6.2 we present a method for reasoning on primitive $\mathcal{LUNI}$-schemata. A complexity analysis shows that the method works in worst case exponential time, implying its optimality from a complexity theoretic point of view. In Section 6.3 we analyze in some more detail issues related to the complexity of the proposed method, and show that in several relevant cases it can be applied efficiently. Finally, in Section 6.4 we show how the techniques can be extended in order to deal with free $\mathcal{LCNI}$-schemata.

6.1 Systems of Linear Inequalities

We regard a system of linear inequalities as a pair $\Psi := (\mathcal{V}, \mathcal{D})$, where $\mathcal{V}$ is the set of unknowns, and $\mathcal{D}$ is the set of linear inequalities.

The following lemma establishes a property of certain systems of inequalities that relates the existence of integer solutions to the existence of arbitrary positive solutions.

Definition 6.1.1 A linear inequality $\sum_{i=1}^{n} c_i \cdot x_i \geq b$ is said to be *positive* if $b \geq 0$. ∎

Lemma 6.1.2 *Let $\Psi = (\{x_1, \ldots, x_n\}, \mathcal{D})$ be a system of positive linear inequalities with integer coefficients. If Ψ admits a nonnegative rational solution Sol, then the following holds:*

1. *For every constant $r \geq 1$, $Sol_r := r \cdot Sol$ is a solution of Ψ.*
2. *Ψ admits a nonnegative integer solution.*

Proof. (1) For a homogeneous inequality $\sum_{i=1}^{n} c_i \cdot x_i \geq 0$ in $\mathcal{D}$ and for any positive constant r we have that

$$\sum_{i=1}^{n} c_i \cdot Sol_r(x_i) = \sum_{i=1}^{n} c_i \cdot r \cdot Sol(x_i) = r \cdot \sum_{i=1}^{n} c_i \cdot Sol(x_i) \geq 0,$$

where the inequality holds since *Sol* is a solution of Ψ. For a positive inequality $\sum_{i=1}^{n} c_i \cdot x_i \geq b$ in $\mathcal{D}$ and for any constant $r \geq 1$ we have that

$$\sum_{i=1}^{n} c_i \cdot Sol_r(x_i) = \sum_{i=1}^{n} c_i \cdot r \cdot Sol(x_i) = r \cdot \sum_{i=1}^{n} c_i \cdot Sol(x_i) \geq r \cdot b \geq b,$$

where the last inequalities hold since *Sol* is a solution of Ψ, $r \geq 1$, and $b > 0$. This shows that Sol_r is a solution of Ψ.

(2) Let r be some common multiplier of the denominators of the rational values assigned by *Sol* to $x_1, \ldots, x_n$, if these values are viewed as integer fractions. Then Sol_r assigns only integer values to the unknowns and by (1) it is a solution of Ψ. □

6.1.1 Acceptable Solutions

We now define a special class of solutions for a system $\Psi := (\mathcal{V}, \mathcal{D})$ of positive linear inequalities with integer coefficients. We will see in the following sections that solutions belonging to such class reflect constraints that arise naturally due to the semantics of $\mathcal{L}$-schemata.

Let the set $\mathcal{V}$ of unknowns be partitioned into two sets $\mathcal{V}_f := \{f_1, \ldots, f_{n_f}\}$ and $\mathcal{V}_d := \{d_1, \ldots, d_{n_d}\}$. We call the unknowns of $\mathcal{V}_f$ *free* and those of $\mathcal{V}_d$ *dependent*. Let further $\mathcal{W} \subseteq \mathcal{V}_d \times \mathcal{V}_f$ be a total relation from $\mathcal{V}_d$ to $\mathcal{V}_f$, i.e. the domain of $\mathcal{W}$ equals $\mathcal{V}_d$.

Definition 6.1.3 A solution *Sol* of Ψ is said to be *$\mathcal{W}$-acceptable* if

$$\forall d_i \in \mathcal{V}_d, \ \forall f_j \in \mathcal{V}_f : \ (d_i, f_j) \in \mathcal{W} \wedge Sol(f_j) = 0 \ \rightarrow \ Sol(d_i) = 0.$$

■

In other words, for a $\mathcal{W}$-acceptable solution *Sol*, an unknown f that appears in the codomain of $\mathcal{W}$ and that is assigned value 0 by *Sol*, forces all unknowns of which f is a $\mathcal{W}$-image to be also assigned value 0 by *Sol*. To give an intuitive idea of why such a condition is needed, we anticipate that the system of inequalities we construct has unknowns corresponding to classes and attributes, and that a solution of the system of inequalities gives the cardinalities of the extensions of classes and attributes in a possible model of the schema. An acceptable solution ensures that if an unknown is assigned value 0 and therefore the corresponding class has an empty extension, then also all attributes that refer to that class have an empty extension.

The following lemma suggests an algorithm for verifying the existence of $\mathcal{W}$-acceptable nonnegative integer solutions of Ψ that works in nondeterministic polynomial time in the size of Ψ.

Lemma 6.1.4 *For a subset $Z \subseteq \mathcal{V}_f$ of the free unknowns, let $\Psi_Z := (\mathcal{V}, \mathcal{D}_Z)$ be the following system of inequalities:*

$$\begin{aligned} \mathcal{D}_Z \ := \ \ \mathcal{D} \ & \cup \ \{f_j = 0 \mid f_j \in Z\} \\ & \cup \ \{f_j \geq 1 \mid f_j \in \mathcal{V}_f \setminus Z\} \\ & \cup \ \{d_i = 0 \mid d_i \in \mathcal{V}_d \ \wedge \ \exists f_j \in Z : (d_i, f_j) \in \mathcal{W}\}. \end{aligned}$$

Then Ψ admits a $\mathcal{W}$-acceptable nonnegative integer solution if and only if there is a $Z \in \mathcal{V}_f$ such that Ψ_Z admits a nonnegative integer solution.

Proof. "⇐" Since $\mathcal{D} \subseteq \mathcal{D}_Z$, every solution of Ψ_Z is also a solution of Ψ. The claim then follows by observing that Ψ_Z is defined in such a way that all its solutions are $\mathcal{W}$-acceptable.

"⇒" Suppose that Ψ admits a $\mathcal{W}$-acceptable nonnegative integer solution *Sol*. Let $Z := \{f_j \in \mathcal{V}_f \mid Sol(f_j) = 0\}$. Since *Sol* is $\mathcal{W}$-acceptable, for all unknowns $d_i \in \mathcal{V}_d$ such that there is an $f_j \in Z$ with $(d_i, f_j) \in \mathcal{W}$ it holds that $Sol(d_i) = 0$. Therefore *Sol* is also a nonnegative integer solution of Ψ_Z. □

The algorithm to test the existence of $\mathcal{W}$-acceptable nonnegative integer solutions of Ψ works as follows: It selects nondeterministically a subset $Z \in \mathcal{V}_f$ and tests whether Ψ_Z admits a nonnegative solution. If it finds such a Z, it answers "yes", otherwise it answers "no". The correctness of the negative answer is guaranteed by the "only-if" part of Lemma 6.1.4. The correctness of the positive answer is guaranteed by the "if" part of Lemma 6.1.4 by observing that since all inequalities both in Ψ and in Ψ_Z are positive, by Lemma 6.1.2 the existence of arbitrary rational solutions guarantees the existence of integer solutions. Moreover the algorithm works in nondeterministic polynomial time since for all $Z \in \mathcal{V}_f$ the size of Ψ_Z is polynomial in the size of Ψ and the existence of rational solutions of Ψ_Z can be verified in polynomial time in the size of Ψ_Z (see for example [145]).

However, by exploiting the previous result and making use of the following lemma we can show that the existence of $\mathcal{W}$-acceptable nonnegative integer solutions of Ψ can be verified in (deterministic) polynomial time in the size of Ψ. The lemma is an easy consequence of a well-known result about the complexity of integer programming (see for example [123]) and which essentially states that the existence of integer solutions of a system of inequalities implies the existence of solutions of bounded size.

Lemma 6.1.5 *Let Ψ be a system of m linear inequalities in n unknowns, and let the coefficients and constants that appear in the inequalities be in $\{0, \pm 1, \ldots, \pm a\}$. If Ψ admits a nonnegative integer solution, then it also admits one in which the values assigned to the unknowns are all bounded by*

$$H(\Psi) := (n + m) \cdot (m \cdot a)^{2m+1}.$$

Proof. Let $\Psi := (\mathcal{V}, \mathcal{D})$. Without loss of generality we can assume that Ψ has the following form:

$$\begin{aligned} \mathcal{V} &:= \{x_1, \ldots, x_n\} \\ \mathcal{D} &:= \left\{ \sum_{i=1}^{n} c_{ij} \cdot x_i \geq b_j \;\middle|\; j \in \{1, \ldots, m\} \right\}. \end{aligned}$$

Let further $\Psi' := (\mathcal{V}', \mathcal{D}')$ be the following system of linear equations

$$\begin{aligned} \mathcal{V}' &:= \{x_1, \ldots, x_n, y_1, \ldots, y_m\} \\ \mathcal{D}' &:= \left\{ \sum_{i=1}^{n} c_{ij} \cdot x_i - y_j = b_j \;\middle|\; j \in \{1, \ldots, m\} \right\}, \end{aligned}$$

where the unknowns $y_1, \ldots, y_m$ are not present in $\mathcal{V}$. Clearly every nonnegative solution of Ψ' is also a solution of Ψ, if restricted to the unknowns in $\mathcal{V}$, and every nonnegative solution of Ψ can be extended to a nonnegative solution of Ψ'. The claim then follows from the following result proved in [123]: If Ψ' admits a nonnegative solution then it also admits one in which the values assigned to the unknowns are all bounded by $n' \cdot (m \cdot a)^{2m+1}$, where $n' := n + m$ is the number of unknowns in Ψ'. □

Proposition 6.1.6 *Checking if a system Ψ of positive linear inequalities with integer coefficients admits a $\mathcal{W}$-acceptable nonnegative integer solution can be done in polynomial time with respect to $|\Psi|$.*

Proof. By part 2 of Lemma 6.1.2, it is sufficient to check if Ψ admits an $\mathcal{W}$-acceptable nonnegative rational solution. We denote with $SOL(\Psi)$ the set of solutions of Ψ, and with $SOL_{acc}(\Psi)$ the set of $\mathcal{W}$-acceptable solutions. For a positive integer k we define the set $Acc(\Psi, \mathcal{W}, k)$ of inequalities as follows:

$$Acc(\Psi, \mathcal{W}, k) := \{d_i \leq k \cdot f_j \mid d_i \in V_d \ \wedge \ f_j \in V_f \ \wedge \ (d_i, f_j) \in \mathcal{W}\}.$$

For simplicity we denote with $\Psi \cup Acc(\Psi, \mathcal{W}, k)$ the system $(\mathcal{V}, \mathcal{D} \cup Acc(\Psi, \mathcal{W}, k))$. Clearly $\Psi \cup Acc(\Psi, \mathcal{W}, k)$ admits only $\mathcal{W}$-acceptable solutions. Moreover, since all unknowns in $Acc(\Psi, \mathcal{W}, k)$ are also unknowns in Ψ, for every positive integer k the following holds:

$$SOL(\Psi \cup Acc(\Psi, \mathcal{W}, k)) = SOL_{acc}(\Psi \cup Acc(\Psi, \mathcal{W}, k)) \subseteq SOL_{acc}(\Psi).$$

Therefore, if for some positive integer k, $\Psi \cup Acc(\Psi, \mathcal{W}, k)$ admits a solution, this will also be an acceptable solution of Ψ. We show that the converse is also true, in the sense that it is possible to find a value k_Ψ whose binary representation is polynomial in n and m such that the following holds: If Ψ admits an acceptable solution then $\Psi \cup Acc(\Psi, \mathcal{W}, k_\Psi)$ admits a solution. Since the number of inequalities in $Acc(\Psi, \mathcal{W}, k_\Psi)$ is polynomial in n and m and since verifying the existence of solutions of a system of linear inequalities can be done in polynomial time, we are done.

In the rest of the proof we show that such a positive integer k_Ψ can effectively be computed. For $Z \subseteq \mathcal{V}_f$, let Ψ_Z be defined as in Lemma 6.1.4. We show that, if for some $Z \subseteq V_f$, Ψ_Z admits a nonnegative integer solution, and therefore by Lemma 6.1.5 a solution Sol bounded by $h_Z := H(\Psi_Z)$, then Sol is also a solution of $\Psi \cup Acc(\Psi, \mathcal{W}, h_Z)$. In fact, since all solutions of Ψ_Z are solutions of Ψ, the only inequalities that Sol could violate are those of $Acc(\Psi, \mathcal{W}, h_Z)$. Assume by contradiction that Sol violates an inequality $d_i \leq h_Z \cdot f_j$ for some $d_i \in V_d$ and $f_j \in V_f$ with $(d_i, f_j) \in \mathcal{W}$. Since Sol is a solution of Ψ_Z, if $f_j \in Z$ then $Sol(f_j) = 0$ and $Sol(d_i) = 0$, and the inequality is satisfied. If $f_j \notin Z$ then $Sol(f_j) \geq 1$, and since the maximum value that Sol can assign to d_j is h_Z, the inequality is also satisfied. In both cases we obtain a contradiction.

By Lemma 6.1.4 Ψ admits a $\mathcal{W}$-acceptable nonnegative integer solution only if for some $Z \in \mathcal{V}_f$, Ψ_Z admits a nonnegative integer solution. By the above argument this is only the case if $\Psi \cup Acc(\Psi, \mathcal{W}, h_m)$ admits a nonnegative integer solution, where $h_m := \max\{h_Z \mid Z \subseteq V_f\}$. An upper bound for h_m can easily be obtained from Lemma 6.1.5 by observing that Ψ_Z contains not more unknowns than Ψ, and at most $m + |\mathcal{V}_f \setminus Z| < m + n$ inequalities. Therefore it suffices to define

$$k_\Psi := (n + m) \cdot (M \cdot a)^{2M+1} > h_Z,$$

where $M := m + n$, which also guarantees that the binary representation of k_Ψ is polynomial in $|\Psi|$. □

6.2 Finite Model Reasoning on Primitive $\mathcal{LUNI}$-Schemata

In this section we present a technique for reasoning on primitive $\mathcal{LUNI}$-schemata[1] with respect to finite models. The method is based on the idea introduced in [111] to construct from a schema $\mathcal{S}$ a system $\Psi_\mathcal{S}$ of linear inequalities and relate the existence of solutions of $\Psi_\mathcal{S}$ to the existence of models of schema $\mathcal{S}$. The unknowns introduced in $\Psi_\mathcal{S}$ are intended to represent the number of instances of each class and each attribute in a possible finite model of $\mathcal{S}$, while the inequalities take into account the constraints on the number of instances deriving from number restrictions in $\mathcal{S}$. The approach introduced in [111], which is to use one unknown for each class and attribute, works however only for a relatively simple data model, with no possibility to specify inclusions between classes and in which all classes are therefore assumed to be disjoint. Since in $\mathcal{LUNI}$-schemata classes may be forced to have instances in common, it is not possible to proceed in this way. We overcome this problem by introducing the notion of expansion of a schema, from which the system of inequalities can directly be constructed.

We use the following schema as our running example.

Example 6.2.1 Let $\mathcal{S}_b := (\mathcal{C}_b, \mathcal{A}_b, \mathcal{T}_b)$ be the $\mathcal{LUNI}$-schema where

$\mathcal{C}_b := \{\texttt{Root}, \texttt{OtherNode}\}$,
$\mathcal{A}_b := \{\texttt{edge}\}$,

and the set $\mathcal{T}_b$ of assertions consists of:

$$\begin{aligned} \texttt{Root} &\doteq \exists^{\leq 0}\texttt{edge}^- \sqcap \forall \texttt{edge}.\texttt{OtherNode} \sqcap \exists^{\geq 1}\texttt{edge} \sqcap \exists^{\leq 2}\texttt{edge} \\ \texttt{OtherNode} &\doteq \forall \texttt{edge}^-.(\texttt{Root} \sqcup \texttt{OtherNode}) \sqcap \exists^{=1}\texttt{edge}^- \sqcap \\ &\quad \forall \texttt{edge}.\texttt{OtherNode} \sqcap \exists^{\geq 1}\texttt{edge} \sqcap \exists^{\leq 2}\texttt{edge} \sqcap \neg \texttt{Root}. \end{aligned}$$

[1] The types of constructors and assertions allowed in primitive $\mathcal{LUNI}$-schemata are shown in Table 5.1.

The schema $\mathcal{S}_b$ describes the properties of binary trees, where each node has at most one predecessor and at most two successors. In fact, we distinguish between nodes that have no predecessor (`Root`) and those that have one (`OtherNode`). Additionally, we require that each node has at least one successor[2]. This combination of requirements makes the class `Root` finitely inconsistent. The intuitive explanation for this is exactly the same as in Example 2.4.2. The class `OtherNode`, on the other hand, can be populated in a finite model of the schema, although the model we obtain is in a certain sense counterintuitive, since it necessarily contains a cycle in the `edge` relation. The reasoning procedure we are going to develop will confirm this informal argumentation. ∎

6.2.1 Normalization of a Schema

In order to apply our decision procedure we need to perform a preliminary transformation on the schema, which is similar to the one defined in Section 4.3 to eliminate the nesting of constructors in class expressions. We use the term *class literal* to denote a class name or a class name preceded by the symbol "¬". In the sequel the letter D ranges over class literals. An $\mathcal{LUNI}$-class expression is called *normalized* if it is of the form:

$$D \mid D_1 \sqcup D_2 \mid \forall L.D \mid \exists^{\geq m} L \mid \exists^{\leq n} L,$$

where m is a positive and n a nonnegative integer. The definition of normalized schema is identical to the one in Section 4.3, and we repeat it here for completeness. An assertion is said to be *normalized*, if the class expression on the right-hand side is normalized. A primitive $\mathcal{LUNI}$-schema is *normalized* if it is constituted solely by normalized primitive class specifications.

Lemma 6.2.2 *Every primitive $\mathcal{LUNI}$-schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ can be transformed in linear time into an equivalent normalized primitive $\mathcal{LUNI}$-schema $\mathcal{S}' := (\mathcal{C}', \mathcal{A}, \mathcal{T}')$, where $\mathcal{C} \subseteq \mathcal{C}'$.*

Proof. The proof is analogous to the proof of Lemma 4.3.1. □

Example 6.2.1 (cont.) The normalized $\mathcal{LUNI}$-schema $\mathcal{S}_n := (\mathcal{C}_n, \mathcal{A}_n, \mathcal{T}_n)$ corresponding to schema $\mathcal{S}_b$ is given by

$\mathcal{C}_n := \{\texttt{Root}, \texttt{OtherNode}, \texttt{Node}\}$,
$\mathcal{A}_n := \{\texttt{edge}\}$,

and the set $\mathcal{T}_n$ of assertions consists of:

$$\begin{array}{lcl}
\texttt{Node} & \dot{\preceq} & \texttt{Root} \sqcup \texttt{OtherNode} \\
\\
\texttt{Root} & \dot{\preceq} & \exists^{\leq 0}\texttt{edge}^- \\
\texttt{Root} & \dot{\preceq} & \forall \texttt{edge}.\texttt{OtherNode} \\
\texttt{Root} & \dot{\preceq} & \exists^{\geq 1}\texttt{edge} \\
\texttt{Root} & \dot{\preceq} & \exists^{\leq 2}\texttt{edge}
\end{array}
\qquad
\begin{array}{lcl}
\texttt{OtherNode} & \dot{\preceq} & \forall \texttt{edge}^-.\texttt{Node} \\
\texttt{OtherNode} & \dot{\preceq} & \exists^{\geq 1}\texttt{edge}^- \\
\texttt{OtherNode} & \dot{\preceq} & \exists^{\leq 1}\texttt{edge}^- \\
\texttt{OtherNode} & \dot{\preceq} & \forall \texttt{edge}.\texttt{OtherNode} \\
\texttt{OtherNode} & \dot{\preceq} & \exists^{\geq 1}\texttt{edge} \\
\texttt{OtherNode} & \dot{\preceq} & \exists^{\leq 2}\texttt{edge} \\
\texttt{OtherNode} & \dot{\preceq} & \neg \texttt{Root}.
\end{array}$$

Notice that we have introduced an additional class `Node` to replace the disjunction `Root` ⊔ `OtherNode` nested within the universal quantification. ∎

Lemmata 6.2.2 and 2.3.7 allow us to restrict our attention to normalized schemata when devising procedures to perform the reasoning services. Therefore, in the rest of the section we assume that the schemata we deal with are all normalized primitive $\mathcal{LUNI}$-schemata, and that $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$.

[2]The form of the schema was chosen in order to illustrate the reasoning technique, and we are aware of the fact that schema $\mathcal{S}_b$ is not the best way of modeling the desired type of tree

6.2.2 Expansion of a Schema

We now define an alternative representation of a schema from which the system of linear inequalities can be directly derived. The structure of class expressions and of the assertions in this alternative representation is simplified, at the cost of an exponential increase in size.

In the following we call a set of class names a *compound class*, and an element of $\mathcal{A}\times 2^{\mathcal{C}}\times 2^{\mathcal{C}}$ a *compound attribute*. The reason for using this terminology will become clear in the following. $\widehat{F}$ and $\widehat{P}$ range over compound classes and compound attributes respectively. We also use $A[\widehat{F}_1, \widehat{F}_2]$ to denote a compound attribute $(A, \widehat{F}_1, \widehat{F}_2) \in \mathcal{A}\times 2^{\mathcal{C}}\times 2^{\mathcal{C}}$, and we say that the compound attribute *corresponds* to attribute A.

Definition 6.2.3 An *expanded schema* is a tuple $\widehat{\mathcal{S}} = (\mathcal{C}, \mathcal{A}, \widehat{\mathcal{F}}, \widehat{\mathcal{P}}, \widehat{\mathcal{T}})$, where

- $\mathcal{C}$ is a set of class names.
- $\mathcal{A}$ is a set of attribute names.
- $\widehat{\mathcal{F}} \subseteq 2^{\mathcal{C}}$ is a set of compound classes.
- $\widehat{\mathcal{P}} \subseteq \mathcal{A}\times\widehat{\mathcal{F}}\times\widehat{\mathcal{F}}$ is a set of compound attributes.
- $\widehat{\mathcal{T}}$ is a set of *compound assertions*. Each such assertion has one of the forms

$$\begin{array}{lcll} \widehat{F} & \dot{\preceq} & \exists^{\geq m} L, & \text{or} \\ \widehat{F} & \dot{\preceq} & \exists^{\leq n} L, & \end{array}$$

  where $\widehat{F} \in \widehat{\mathcal{F}}$, L is a link, i.e. either an attribute name in $\mathcal{A}$ or the inverse of an attribute name, m is a positive, and n a nonnegative integer. ∎

Where not specified otherwise, we assume $\widehat{\mathcal{S}} = (\mathcal{C}, \mathcal{A}, \widehat{\mathcal{F}}, \widehat{\mathcal{P}}, \widehat{\mathcal{T}})$.

Intuitively, the instances of a compound class $\widehat{F}$ are all those objects of the domain that are instances of all classes in $\widehat{F}$ and are not instances of any class not in $\widehat{F}$. A compound attribute $A[\widehat{F}_1, \widehat{F}_2]$ is interpreted as the restriction of attribute A to the pairs whose first component is an instance of $\widehat{F}_1$ and whose second component is an instance of $\widehat{F}_2$.

More formally, the semantics of an expanded schema $\widehat{\mathcal{S}}$ is given by extending interpretations over $\mathcal{C}$ and $\mathcal{A}$ to compound classes and attributes in the following way:

$$\begin{array}{rcl} \widehat{F}^{\mathcal{I}} & := & \bigcap_{C\in\widehat{F}} C^{\mathcal{I}} \setminus \bigcup_{C\in(\mathcal{C}\setminus\widehat{F})} C^{\mathcal{I}} \\ (A[\widehat{F}_1, \widehat{F}_2])^{\mathcal{I}} & := & A^{\mathcal{I}} \cap (\widehat{F}_1^{\mathcal{I}}\times\widehat{F}_2^{\mathcal{I}}). \end{array}$$

Note that according to this definition two different compound classes have necessarily disjoint interpretations. The same observation holds for two different compound attributes $A[\widehat{F}_1, \widehat{F}_2]$ and $A[\widehat{F}_1', \widehat{F}_2']$ that correspond to the same attribute A.

The following easy lemma, shows how to obtain the extensions of classes and attributes, given the extensions of all compound classes and compound attributes.

Lemma 6.2.4 *Let $C \in \mathcal{C}$ be a class name and $A \in \mathcal{A}$ be an attribute name. Then the following holds:*

$$\begin{array}{rcl} C^{\mathcal{I}} & = & \bigcup_{\widehat{F}\in 2^{\mathcal{C}} \mid C\in\widehat{F}} \widehat{F}^{\mathcal{I}} \\ A^{\mathcal{I}} & = & \bigcup_{\widehat{P}\in\{A\}\times 2^{\mathcal{C}}\times 2^{\mathcal{C}}} \widehat{P}^{\mathcal{I}}. \end{array}$$

Proof. The claim follows directly from the semantics of compound classes and attributes. □

A (finite) interpretation $\mathcal{I} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *satisfies* a compound assertion $\widehat{F} \doteq E$, if $\widehat{F}^{\mathcal{I}} \subseteq E^{\mathcal{I}}$. $\mathcal{I}$ is a *(finite) model* of $\widehat{\mathcal{S}}$ if the following conditions hold:

- For each compound class $\widehat{F} \in 2^{\mathcal{C}} \setminus \widehat{\mathcal{F}}$, it holds that $\widehat{F}^{\mathcal{I}} = \emptyset$.
- For each compound attribute $\widehat{P} \in (\mathcal{A} \times 2^{\mathcal{C}} \times 2^{\mathcal{C}}) \setminus \widehat{\mathcal{P}}$, it holds that $\widehat{P}^{\mathcal{I}} = \emptyset$.
- $\mathcal{I}$ satisfies all compound assertions in $\widehat{\mathcal{T}}$.

The following definition associates to each schema a particular expanded schema. In order to abbreviate definitions and proofs, we use the following convention: Let $\widehat{F} \subseteq 2^{\mathcal{C}}$ be a compound class and D be a class literal. If D is a positive class literal C, then $D \in \widehat{F}$ means simply $C \in \widehat{F}$, and if D is a negative class literal $\neg C$, then we use $D \in \widehat{F}$ to denote $C \notin \widehat{F}$.

Definition 6.2.5 An expanded schema $\widehat{\mathcal{S}} := (\mathcal{C}, \mathcal{A}, \widehat{\mathcal{F}}, \widehat{\mathcal{P}}, \widehat{\mathcal{T}})$ is called the *expansion* of a schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ if the following conditions hold:

- $\widehat{\mathcal{F}}$ is the set of all $\mathcal{S}$-*consistent* compound classes, where a compound class $\widehat{F} \in 2^{\mathcal{C}}$ is said to be $\mathcal{S}$-consistent, if
  - for every $C \in \widehat{F}$, if $(C \doteq D) \in \mathcal{T}$, then $D \in \widehat{F}$.
  - for every $C \in \widehat{F}$, if $(C \doteq D_1 \sqcup D_2) \in \mathcal{T}$, then $D_1 \in \widehat{F}$ or $D_2 \in \widehat{F}$.
- $\widehat{\mathcal{P}}$ is the set of all $\mathcal{S}$-*consistent* compound attributes, where a compound attribute $A[\widehat{F}_1, \widehat{F}_2] \in \mathcal{A} \times 2^{\mathcal{C}} \times 2^{\mathcal{C}}$ is said to be $\mathcal{S}$-consistent, if
  - $\widehat{F}_1$ and $\widehat{F}_2$ are $\mathcal{S}$-consistent.
  - for every $C \in \widehat{F}_1$, if $(C \doteq \forall A.D) \in \mathcal{T}$, then $D \in \widehat{F}_2$.
  - for every $C \in \widehat{F}_2$, if $(C \doteq \forall A^{-}.D) \in \mathcal{T}$, then $D \in \widehat{F}_1$.
- $\widehat{\mathcal{T}}$ is the smallest set of compound assertions such that for every $\widehat{F} \in \widehat{\mathcal{F}}$
  - if for some $C \in \widehat{F}$ there is an assertion $(C \doteq \exists^{\geq m} L) \in \mathcal{T}$, then $\widehat{\mathcal{T}}$ contains the compound assertion $\widehat{F} \doteq \exists^{\geq m_{max}} L$, where
$$m_{max} := \max\{m \mid \exists C \in \widehat{F} \colon (C \doteq \exists^{\geq m} L) \in \mathcal{T}\}.$$
  - if for some $C \in \widehat{F}$ there is an assertion $(C \doteq \exists^{\leq n} L) \in \mathcal{T}$, then $\widehat{\mathcal{T}}$ contains the compound assertion $\widehat{F} \doteq \exists^{\leq n_{min}} L$, where
$$n_{min} := \min\{n \mid \exists C \in \widehat{F} \colon (C \doteq \exists^{\leq n} L) \in \mathcal{T}\}.$$

■

It is easy to see that the size of the expansion is exponential in the size of the original schema. The following lemma shows that it can also be effectively constructed in exponential time.

Lemma 6.2.6 *The expansion $\widehat{\mathcal{S}}$ of $\mathcal{S}$ can be constructed in time which is exponential in $|\mathcal{S}|$.*

Proof. The set $\widehat{\mathcal{F}}$ of $\mathcal{S}$-consistent compound classes can be constructed by enumerating the exponentially many compound classes and compound attributes and discarding those that are not $\mathcal{S}$-consistent. The check if a compound class is consistent is essentially equivalent to the evaluation of a propositional formula (obtained by considering only the propositional part of $\mathcal{S}$) with respect to a propositional truth assignments (corresponding to the compound class to be checked). Therefore it can be performed in time linear in $|\mathcal{S}|$ and the whole construction of $\widehat{\mathcal{F}}$ requires time at most exponential in $|\mathcal{S}|$. A similar argument holds for the set $\widehat{\mathcal{P}}$ of $\mathcal{S}$-consistent compound attributes. The claim then follows by observing that for each compound class in $\widehat{\mathcal{F}}$ there is at most one compound assertion for each assertion in $\mathcal{T}$. □

Example 6.2.1 (cont.) The expansion $\widehat{\mathcal{S}}_n := (\mathcal{C}_n, \mathcal{A}_n, \widehat{\mathcal{F}}_n, \widehat{\mathcal{P}}_n, \widehat{\mathcal{T}}_n)$ of the normalized schema $\mathcal{S}_n$ is given by

$\mathcal{C}_n := \{\texttt{Root}, \texttt{OtherNode Node}\}$,

$\mathcal{A}_n := \{\texttt{edge}\}$,

$\widehat{\mathcal{F}}_n := \{\texttt{E}, \texttt{R}, \texttt{O}, \texttt{RN}, \texttt{ON}\}$, where
$\texttt{E} := \emptyset, \quad \texttt{R} := \{\texttt{Root}\}, \quad \texttt{O} := \{\texttt{OtherNode}\},$
$\texttt{RN} := \{\texttt{Node}, \texttt{Root}\}, \quad \texttt{ON} := \{\texttt{Node}, \texttt{OtherNode}\},$

$\widehat{\mathcal{P}}_n := \{\texttt{edge[RN,O]}, \texttt{edge[RN,ON]}, \texttt{edge[ON,O]}, \texttt{edge[ON,ON]},$
$\texttt{edge[E,R]}, \texttt{edge[E,RN]}, \texttt{edge[E,E]}\},$

and the set $\widehat{\mathcal{T}}_n$ of compound assertions consists of:

$$\begin{array}{lll} \texttt{R} & \dot{\preceq} & \exists^{\leq 0}\texttt{edge}^- \\ \texttt{R} & \dot{\preceq} & \exists^{\geq 1}\texttt{edge} \\ \texttt{R} & \dot{\preceq} & \exists^{\leq 2}\texttt{edge} \\ \texttt{O} & \dot{\preceq} & \exists^{\geq 1}\texttt{edge}^- \\ \texttt{O} & \dot{\preceq} & \exists^{\leq 1}\texttt{edge}^- \\ \texttt{O} & \dot{\preceq} & \exists^{\geq 1}\texttt{edge} \\ \texttt{O} & \dot{\preceq} & \exists^{\leq 2}\texttt{edge} \end{array} \qquad \begin{array}{lll} \texttt{RN} & \dot{\preceq} & \exists^{\leq 0}\texttt{edge}^- \\ \texttt{RN} & \dot{\preceq} & \exists^{\geq 1}\texttt{edge} \\ \texttt{RN} & \dot{\preceq} & \exists^{\leq 2}\texttt{edge} \\ \texttt{ON} & \dot{\preceq} & \exists^{\geq 1}\texttt{edge}^- \\ \texttt{ON} & \dot{\preceq} & \exists^{\leq 1}\texttt{edge}^- \\ \texttt{ON} & \dot{\preceq} & \exists^{\geq 1}\texttt{edge} \\ \texttt{ON} & \dot{\preceq} & \exists^{\leq 2}\texttt{edge}. \end{array}$$

■

The following lemmata relate the (finite) models of a schema to the (finite) models of its expansion.

Lemma 6.2.7 *Let $\mathcal{S}$ be a schema and $\mathcal{I}$ a (finite) model of $\mathcal{S}$. Then for each $\mathcal{S}$-inconsistent compound class or compound attribute X, $X^{\mathcal{I}} = \emptyset$.*

Proof. Let $\widehat{F} \in 2^{\mathcal{C}}$ be $\mathcal{S}$-inconsistent. Then (a) there is an assertion $(C \dot{\preceq} D) \in \mathcal{T}$ such that $C \in \widehat{F}$ and $D \notin \widehat{F}$, or (b) there is an assertion $(C \dot{\preceq} D_1 \sqcup D_2) \in \mathcal{T}$ such that $C \in \widehat{F}$, $D_1 \notin \widehat{F}$, and $D_2 \notin \widehat{F}$. In case (a) $\widehat{F}^{\mathcal{I}} \subseteq C^{\mathcal{I}} \setminus D^{\mathcal{I}}$, and since $\mathcal{I}$ satisfies $C \dot{\preceq} D$, $\widehat{F}^{\mathcal{I}} = \emptyset$. In case (b) $\widehat{F}^{\mathcal{I}} \subseteq C^{\mathcal{I}} \setminus (D_1^{\mathcal{I}} \cup D_2^{\mathcal{I}})$, and since $\mathcal{I}$ satisfies $C \dot{\preceq} D_1 \sqcup D_2$, $\widehat{F}^{\mathcal{I}} = \emptyset$.

Let $\widehat{P} = A[\widehat{F}_1, \widehat{F}_2] \in \mathcal{A} \times 2^{\mathcal{C}} \times 2^{\mathcal{C}}$ be $\mathcal{S}$-inconsistent. Then (a) $\widehat{F}_1$ or $\widehat{F}_2$ is $\mathcal{S}$-inconsistent, or (b) there is an assertion $(C \dot{\preceq} \forall A.D) \in \mathcal{T}$ such that $C \in \widehat{F}_1$ and $D \notin \widehat{F}_2$, or (c) there is an assertion $(C \dot{\preceq} \forall A^-.D) \in \mathcal{T}$ such that $C \in \widehat{F}_2$ and $D \notin \widehat{F}_1$. In case (a) $\widehat{F}_1^{\mathcal{I}} = \emptyset$ or $\widehat{F}_2^{\mathcal{I}} = \emptyset$, and therefore $\widehat{P}^{\mathcal{I}} \subseteq \widehat{F}_1^{\mathcal{I}} \times \widehat{F}_2^{\mathcal{I}} = \emptyset$. In case (b), assume by contradiction that $(o_1, o_2) \in \widehat{P}^{\mathcal{I}} \subseteq A^{\mathcal{I}}$ for some $o_1, o_2 \in \Delta^{\mathcal{I}}$. Then $o_1 \in \widehat{F}_1^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ and $o_2 \in \widehat{F}_2^{\mathcal{I}}$. Since $\mathcal{I}$ satisfies $C \dot{\preceq} \forall A.D$, it follows that $o_2 \in D^{\mathcal{I}}$. But $\widehat{F}_2^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ gives rise to a contradiction. Case (c) can be treated in a similar way. □

Lemma 6.2.8 *Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be a schema, $\widehat{\mathcal{S}}$ its expansion, and $\mathcal{I}$ a (finite) interpretation over $\mathcal{C}$ and $\mathcal{A}$. Then $\mathcal{I}$ is a (finite) model of $\mathcal{S}$ if and only if it is a (finite) model of $\widehat{\mathcal{S}}$.*

Proof. "⇒" Let $\mathcal{I}$ be a (finite) model of $\mathcal{S}$. Since $\widehat{\mathcal{F}}$ contains exactly the set of $\mathcal{S}$-consistent compound classes, by Lemma 6.2.7, for a compound class $\widehat{F} \in 2^{\mathcal{C}} \setminus \widehat{\mathcal{F}}$ we have that $\widehat{F}^{\mathcal{I}} = \emptyset$. The same argument holds for compound attributes. Let $(\widehat{F} \dot{\preceq} \exists^{\geq m} L) \in \widehat{\mathcal{T}}$. Then there is a $C \in \widehat{F}$ such that $(C \dot{\preceq} \exists^{\geq m} L) \in \mathcal{T}$. Since $\mathcal{I}$ satisfies this assertion and $\widehat{F}^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, $\mathcal{I}$ satisfies also $\widehat{F} \dot{\preceq} \exists^{\geq m} L$. A similar argument holds if $(\widehat{F} \dot{\preceq} \exists^{\leq n} L) \in \widehat{\mathcal{T}}$.

"⇐" Let $\mathcal{I}$ be a (finite) model of $\widehat{\mathcal{S}}$ and assume by contradiction that there is an assertion $(C \dot{\preceq} E) \in \mathcal{T}$ not satisfied by $\mathcal{I}$. This means there is some $o \in \Delta^{\mathcal{I}}$ such that $o \in C^{\mathcal{I}}$ and $o \notin E^{\mathcal{I}}$. Let $\widehat{F}_o$ be the (unique) compound class such that $o \in \widehat{F}_o^{\mathcal{I}}$. Then we have that $C \in \widehat{F}_o$. Suppose $E = D$. Since $o \notin D^{\mathcal{I}}$, we have that $D \notin \widehat{F}_o$. It follows that $\widehat{F}_o$ is not $\mathcal{S}$-consistent, which by Lemma 6.2.7 contradicts $o \in \widehat{F}_o^{\mathcal{I}}$. The case where $E = D_1 \sqcup D_2$ can be handled in a similar way. Suppose $E = \forall L.D$. Since $o \notin (\forall A.D)^{\mathcal{I}}$ there is some $o' \in \Delta^{\mathcal{I}}$ such that $(o, o') \in A^{\mathcal{I}}$ and $o' \notin D^{\mathcal{I}}$. Let $\widehat{F}_{o'}$ be the (unique) compound class such that $o' \in \widehat{F}_{o'}^{\mathcal{I}}$.

Then $D \notin \widehat{F}_{o'}$ and therefore $A[\widehat{F}_o, \widehat{F}_{o'}]$ is not consistent. By Lemma 6.2.7 this contradicts $(o, o') \in A^{\mathcal{I}}$, $a \in \widehat{F}_o^{\mathcal{I}}$, and $o' \in \widehat{F}_{o'}^{\mathcal{I}}$. The case where $E = \forall A^-.D$ can be handled in a similar way. Suppose $E = \exists^{\geq m} L$. Since $C \in \widehat{F}_o$, $\widehat{\mathcal{T}}$ contains an assertion $\widehat{F}_o \doteq \exists^{\geq m_{max}} L$, where $m_{max} \geq m$. From $o \notin (\exists^{\geq m} L)^{\mathcal{I}}$ it follows that $o \notin (\exists^{\geq m_{max}} L)^{\mathcal{I}}$, which contradicts the fact that $\mathcal{I}$ satisfies all axioms in $\widehat{\mathcal{T}}$. The case where $E = \exists^{\leq n} L$ can be handled in a similar way. □

6.2.3 System of Inequalities Corresponding to a Schema

We are now ready to define a system of linear inequalities whose solutions of a certain type are related to the finite models of the schema.

Definition 6.2.9 Let $\mathcal{S}$ be a schema and $\widehat{\mathcal{S}} := (\mathcal{C}, \mathcal{A}, \widehat{\mathcal{F}}, \widehat{\mathcal{P}}, \widehat{\mathcal{T}})$ its expansion. Then the system $\Psi_{\mathcal{S}} := (\mathcal{V}, \mathcal{D})$ *corresponding to* $\mathcal{S}$ is defined as follows:

- $\mathcal{V} := \mathcal{V}_{\widehat{\mathcal{F}}} \cup \mathcal{V}_{\widehat{\mathcal{P}}}$ is a set of unknowns, where
  - $\mathcal{V}_{\widehat{\mathcal{F}}} := \{\mathrm{Var}(\widehat{F}) \mid \widehat{F} \in \widehat{\mathcal{F}}\}$ and
  - $\mathcal{V}_{\widehat{\mathcal{P}}} := \{\mathrm{Var}(\widehat{P}) \mid \widehat{P} \in \widehat{\mathcal{P}}\}$.
- $\mathcal{D}$ is a set of homogeneous linear inequalities over the unknowns in $\mathcal{V}$ constituted by the following inequalities:
  - For each compound assertion $(\widehat{F} \doteq \exists^{\geq m} L) \in \widehat{\mathcal{T}}$ the inequality

$$m \cdot \mathrm{Var}(\widehat{F}) \leq S(\widehat{F}, L), \tag{6.1}$$

    where

$$\begin{aligned} S(\widehat{F}, A) &:= \sum_{A[\widehat{F}, \widehat{F}_2] \in \widehat{\mathcal{F}}} \mathrm{Var}(A[\widehat{F}, \widehat{F}_2]) \\ S(\widehat{F}, A^-) &:= \sum_{A[\widehat{F}_1, \widehat{F}] \in \widehat{\mathcal{F}}} \mathrm{Var}(A[\widehat{F}_1, \widehat{F}]). \end{aligned}$$

  - For each compound assertion $(\widehat{F} \doteq \exists^{\leq n} L) \in \widehat{\mathcal{T}}$ the inequality

$$n \cdot \mathrm{Var}(\widehat{F}) \geq S(\widehat{F}, L). \tag{6.2}$$

■

Where not specified otherwise we assume $\Psi_{\mathcal{S}} = (\mathcal{V}, \mathcal{D})$. The following lemma gives an exponential upper bound for the construction of $\Psi_{\mathcal{S}}$.

Lemma 6.2.10 *The system of inequalities $\Psi_{\mathcal{S}}$ can be constructed in time which is at most exponential in $|\mathcal{S}|$.*

Proof. By Lemma 6.2.6 the expansion $\widehat{\mathcal{S}}$ of $\mathcal{S}$ can be constructed in exponential time in $|\mathcal{S}|$, and therefore it has also at most exponential size in $|\mathcal{S}|$. $\Psi_{\mathcal{S}}$ contains at most one inequality for each compound assertion in $\widehat{\mathcal{S}}$, and each such inequality contains a number of unknowns that is smaller than the number of compound classes and compound attributes together. Therefore $|\Psi_{\mathcal{S}}|$ is at most quadratic in $|\widehat{\mathcal{S}}|$ and it can also be effectively constructed in time that is quadratic in $|\widehat{\mathcal{S}}|$. Summing up we obtain that $\Psi_{\mathcal{S}}$ can be constructed in time at most exponential in $|\mathcal{S}|$. □

Example 6.2.1 (cont.) The system $\Psi_{\mathcal{S}_n} := (\mathcal{V}_n, \mathcal{D}_n)$ derived from the expansion $\widehat{\mathcal{S}}_n$ is defined as follows:
$\mathcal{V}_n := \{\mathtt{e}, \mathtt{r}, \mathtt{o}, \mathtt{rn}, \mathtt{on}, \mathtt{e}_{\mathtt{RN,O}}, \mathtt{e}_{\mathtt{RN,ON}}, \mathtt{e}_{\mathtt{ON,O}}, \mathtt{e}_{\mathtt{ON,ON}}, \mathtt{e}_{\mathtt{E,R}}, \mathtt{e}_{\mathtt{E,RN}}, \mathtt{e}_{\mathtt{E,E}}\}$,
where we have denoted the unknown corresponding to a compound class with the name of the compound class in lower-case letters, and the unknown corresponding to a compound attribute $\mathtt{edge}[X, Y]$ with $\mathtt{e}_{X,Y}$. The set $\mathcal{D}_n$ of inequalities consists of:

$$\begin{array}{lcl@{\qquad}lcl}
0{\cdot}\mathtt{r} & \geq & \mathtt{e}_{\mathtt{E,R}} & 0{\cdot}\mathtt{rn} & \geq & \mathtt{e}_{\mathtt{E,RN}} \\
1{\cdot}\mathtt{r} & \leq & 0 & 1{\cdot}\mathtt{rn} & \leq & \mathtt{e}_{\mathtt{RN,O}} + \mathtt{e}_{\mathtt{RN,ON}} \\
2{\cdot}\mathtt{r} & \geq & 0 & 2{\cdot}\mathtt{rn} & \geq & \mathtt{e}_{\mathtt{RN,O}} + \mathtt{e}_{\mathtt{RN,ON}} \\
1{\cdot}\mathtt{o} & \leq & \mathtt{e}_{\mathtt{RN,O}} + \mathtt{e}_{\mathtt{ON,O}} & 1{\cdot}\mathtt{on} & \leq & \mathtt{e}_{\mathtt{RN,ON}} + \mathtt{e}_{\mathtt{ON,ON}} \\
1{\cdot}\mathtt{o} & \geq & \mathtt{e}_{\mathtt{RN,O}} + \mathtt{e}_{\mathtt{ON,O}} & 1{\cdot}\mathtt{on} & \geq & \mathtt{e}_{\mathtt{RN,ON}} + \mathtt{e}_{\mathtt{ON,ON}} \\
1{\cdot}\mathtt{o} & \leq & 0 & 1{\cdot}\mathtt{on} & \leq & \mathtt{e}_{\mathtt{ON,O}} + \mathtt{e}_{\mathtt{ON,ON}} \\
2{\cdot}\mathtt{o} & \geq & 0 & 2{\cdot}\mathtt{on} & \geq & \mathtt{e}_{\mathtt{ON,O}} + \mathtt{e}_{\mathtt{ON,ON}}.
\end{array}$$

After some simplifications we obtain the following equivalent system:

$$\begin{array}{l}
\mathtt{r} = \mathtt{o} = 0 \\
\mathtt{e}_{\mathtt{RN,O}} = \mathtt{e}_{\mathtt{ON,O}} = \mathtt{e}_{\mathtt{E,R}} = \mathtt{e}_{\mathtt{E,RN}} = 0 \\
\mathtt{rn} \leq \mathtt{e}_{\mathtt{RN,ON}} \leq 2{\cdot}\mathtt{rn} \\
\mathtt{on} \leq \mathtt{e}_{\mathtt{ON,ON}} \leq 2{\cdot}\mathtt{on} \\
\mathtt{on} = \mathtt{e}_{\mathtt{RN,ON}} + \mathtt{e}_{\mathtt{ON,ON}}.
\end{array}$$

■

6.2.4 Characterization of Finite Class Consistency

In order to establish a correspondence between the existence of particular solutions of $\Psi_{\mathcal{S}}$ and the existence of finite models of a schema, consider a simple normalized schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$, where $\mathcal{A}$ contains only one attribute name A. We prove a preliminary result which gives a sufficient condition for a finite interpretation $\mathcal{I}$ over $\mathcal{C}$ and $\mathcal{A}$ to be a model of $\mathcal{S}$. Let $\widehat{\mathcal{S}} := (\mathcal{C}, \mathcal{A}, \widehat{\mathcal{F}}, \widehat{\mathcal{P}}, \widehat{\mathcal{T}})$ be the expansion of $\mathcal{S}$ and assume $\widehat{\mathcal{F}} := \{\widehat{F}_1, \ldots, \widehat{F}_K\}$. The compound attribute $A[\widehat{F}_k, \widehat{F}_{k'}]$ is abbreviated with $\widehat{A}_{kk'}$. We also use $\widehat{f}_k$ as an abbreviation for $\mathrm{Var}(\widehat{F}_k)$, and $\widehat{a}_{kk'}$ as an abbreviation for $\mathrm{Var}(\widehat{A}_{kk'})$. With I_F we denote the set $\{1, \ldots, K\}$ of indexes of compound classes in $\widehat{\mathcal{F}}$, and with I_P we denote the set of pairs of indexes (k, k') of compound attributes in $\widehat{\mathcal{P}}$, i.e. $I_P := \{(k, k') \mid \widehat{A}_{kk'} \in \widehat{\mathcal{P}}\}$. Let $\Psi_{\mathcal{S}}$ be the system of linear inequalities obtained from $\widehat{\mathcal{S}}$ as specified in Section 6.2.3.

Lemma 6.2.11 *Let $\bar{\mathcal{F}} := \{\bar{f}_k \mid k \in I_F\}$ and $\bar{\mathcal{P}} := \{\bar{a}_{kk'} \mid (k, k') \in I_P\}$ be sets of nonnegative integer numbers such that $\Psi_{\mathcal{S}}$ is satisfied when we substitute $\bar{f}_k$ for $\widehat{f}_k$ and $\bar{a}_{kk'}$ for $\widehat{a}_{kk'}$. A finite model $\mathcal{I}$ of $\widehat{\mathcal{S}}$ exists where*

1. *$\sharp\widehat{F}_k^{\mathcal{I}} = \bar{f}_k$, for $k \in I_F$, and*
2. *$\sharp\widehat{A}_{kk'}^{\mathcal{I}} = \bar{a}_{kk'}$, for $(k, k') \in I_P$,*

if and only if

$$\bar{a}_{kk'} \leq \bar{f}_k \cdot \bar{f}_{k'}, \quad \textit{for} \quad (k, k') \in I_P. \tag{6.3}$$

Proof. "$\Leftarrow$" Suppose condition 6.3 holds. We exhibit a finite model $\mathcal{I} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\widehat{\mathcal{S}}$ with $\bar{f}_k$ instances of $\widehat{F}_k$, for $k \in I_F$, and $\bar{a}_{kk'}$ instances of $\widehat{A}_{kk'}$, for $(k, k') \in I_P$. For each $k \in I_F$ such that $\bar{f}_k \neq 0$, we introduce $\bar{f}_k$ symbols $o_k^1, \ldots, o_k^{\bar{f}_k}$. Let $\Delta^{\mathcal{I}}$ be the set of all symbols o_k^j. Figure 6.1 specifies how to assign to each compound class $\widehat{F}_k$ a set $\mathrm{Ext}(\widehat{F}_k) \subseteq \Delta^{\mathcal{I}}$ and to A a set $\mathrm{Ext}(A) \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ of pairs of elements of $\Delta^{\mathcal{I}}$. We show that by putting $\widehat{F}_k^{\mathcal{I}} := \mathrm{Ext}(\widehat{F}_k)$ and $\widehat{A}_{kk'}^{\mathcal{I}} := \mathrm{Ext}(A) \cap (\mathrm{Ext}(\widehat{F}_k) \times \mathrm{Ext}(\widehat{F}_{k'}))$ we indeed obtain a finite interpretation which is also a model of $\widehat{\mathcal{S}}$.

In the rest of the proof, when we refer to a step, we implicitly mean a step in the construction of Figure 6.1. In order to show that $\mathcal{I}$ is a model of $\widehat{\mathcal{S}}$, we have to ensure the following:

1. 1. For each compound class $\widehat{F} \in 2^{\mathcal{C}} \setminus \widehat{\mathcal{F}}$, let $\mathrm{Ext}(\widehat{F}) \leftarrow \emptyset$.
   2. For each compound attribute $\widehat{P} \in \mathcal{A} \times 2^{\mathcal{C}} \times 2^{\mathcal{C}} \setminus \widehat{\mathcal{P}}$, let $\mathrm{Ext}(\widehat{P}) \leftarrow \emptyset$.
   3. For each $k \in I_F$, if $\bar{f}_k = 0$, then let $\mathrm{Ext}(\widehat{F}_k) \leftarrow \emptyset$, else let $\mathrm{Ext}(\widehat{F}_k) \leftarrow \{o_k^1, \ldots, o_k^{\bar{f}_k}\}$.
   4. For each $(k, k') \in I_P$, if $\bar{a}_{kk'} = 0$, then $\mathrm{Ext}(\widehat{A}_{kk'}) \leftarrow \emptyset$, else introduce $\bar{a}_{kk'}$ pairs, assign to each such pair p a label $\mathrm{lab}(p) := (k, k')$, and let $\mathrm{Ext}(\widehat{A}_{kk'}) \leftarrow \{p \mid \mathrm{lab}(p) = (k, k')\}$. Let $\mathrm{Ext}(A)$ be the set of all introduced pairs.
2. 1. Sort the elements of $\mathrm{Ext}(A)$ so that
      - elements with different labels are sorted with respect to the usual lexicographic ordering on their labels, and
      - elements with the same label are sorted arbitrarily.
   2. For each $k \in I_F$, if $\bar{f}_k \neq 0$ then
      1. Mark all elements $p \in \mathrm{Ext}(A)$ with $\mathrm{lab}(p) = (k, k')$ for some $k' \in I_F$. Let M be the number of marked elements and let p_m, for $m \in \{1, \ldots, M\}$, denote the m-th marked element.
      2. For $m \leftarrow 1$ to M (for each marked element) do
         assign o_k^i to the first component of p_m, where
         $i := 1 + (m - 1) \bmod \bar{f}_k$.
      3. Un-mark all elements of $\mathrm{Ext}(A)$;
3. 1. Sort the elements of $\mathrm{Ext}(A)$ so that
      - elements with different labels are sorted with respect to the lexicographic ordering on their labels obtained by considering the components of the label in reverse order, and
      - elements with the same label are sorted in such a way that elements with the same first component (according to step 2.2.2) are contiguous.
   2. For each $k' \in I_F$, if $\bar{f}_{k'} \neq 0$ then
      1. Mark all elements $p \in \mathrm{Ext}(A)$ with $\mathrm{lab}(p) = (k, k')$ for some $k \in I_F$. Let M be the number of marked elements and let p_m, for $m \in \{1, \ldots, M\}$, denote the m-th marked element.
      2. For $m \leftarrow 1$ to M (for each marked element) do
         assign $o_k'^i$ to the second component of p_m, where
         $i = 1 + (m - 1) \bmod \bar{f}_{k'}$.
      3. Un-mark all elements of $\mathrm{Ext}(A)$.

Figure 6.1: Construction of a finite model of $\widehat{\mathcal{S}}$

(a) Each pair in $\mathrm{Ext}(A)$ gets assigned an element of $\Delta^{\mathcal{I}}$ to both of its components.

(b) For each compound class $\widehat{F} \in 2^{\mathcal{C}} \setminus \widehat{\mathcal{F}}$, it holds that $\widehat{F}^{\mathcal{I}} = \emptyset$, and for each compound attribute $\widehat{P} \in (\mathcal{A}\times 2^{\mathcal{C}}\times 2^{\mathcal{C}}) \setminus \widehat{\mathcal{P}}$, it holds that $\widehat{P}^{\mathcal{I}} = \emptyset$.

(c) There is no couple of pairs in $\mathrm{Ext}(A)$ that get assigned the same pair of elements of $\Delta^{\mathcal{I}}$ to both components.

(d) $\mathcal{I}$ satisfies all assertions in $\widehat{\mathcal{T}}$.

With respect to case (a), since condition 6.3 holds, if an unknown $\bar{f}_k = 0$, then also $\bar{a}_{kk'} = \bar{a}_{k'k} = 0$ for all $k' \in I_F$ and in step 1.4 no pairs labeled with (k, k') or (k', k) are introduced. This means that each pair in $\mathrm{Ext}(A)$ gets marked exactly once in step 2.2.1 and exactly once in step 3.2.1 and therefore gets assigned in steps 2.2.3 and 3.2.3 an instance of some compound class to the first and the second component.

With respect to case (b), step 1.1 ensures for each compound class $\widehat{F} \in 2^{\mathcal{C}} \setminus \widehat{\mathcal{F}}$, that $\mathrm{Ext}(\widehat{F}) = \emptyset$, and since we have defined $\widehat{F}^{\mathcal{I}} := \mathrm{Ext}(\widehat{F})$, we have $\widehat{F}^{\mathcal{I}} = \emptyset$. Similarly, step 1.2 ensures for each compound attribute $\widehat{P} \in (\mathcal{A}\times 2^{\mathcal{C}}\times 2^{\mathcal{C}}) \setminus \widehat{\mathcal{P}}$, that $\mathrm{Ext}(\widehat{P}) = \emptyset$, and since we have defined $\widehat{P}^{\mathcal{I}} := \mathrm{Ext}(\widehat{P})$, we have that $\widehat{P}^{\mathcal{I}} = \emptyset$.

With respect to case (c), notice that in step 2.2.2 an element f_k^j can be assigned to the first component of a pair p only if $\mathrm{lab}(p) = (k, k')$ for some $k' \in I_F$. A similar observation holds for the second component of p. This means that two pairs of $\mathrm{Ext}(A)$ can never get assigned the same pair of components if their labels are different. Therefore it is sufficient to show that for every $(k, k') \in I_P$, two pairs with the same label (k, k') cannot get assigned the same pair of components. The number of pairs labeled with (k, k') is equal to $\bar{a}_{kk'}$. After the iteration of step 2.2 relative to the index k, the largest group of pairs that get assigned the same element of $\mathrm{Ext}(\widehat{F}_k)$ to their first component has at most

$$\rho_1 = \left\lceil \bar{a}_{kk'} / \bar{f}_k \right\rceil \leq \bar{f}_{k'}$$

elements, where the inequality holds because of condition 6.3. Analogously, after the iteration of step 3.3 relative to the index k', the largest group of pairs that get assigned the same set of elements of $\mathrm{Ext}(\widehat{F}_k)$ and $\mathrm{Ext}(\widehat{F}_{k'})$ to the first and second components respectively, has at most

$$\rho_2 = \left\lceil \rho_1 / \bar{f}_{k'} \right\rceil \leq 1$$

elements.

With respect to case (d), consider an assertion $(\widehat{F}_k \dot{\preceq} \exists^{\geq m} A) \in \widehat{\mathcal{T}}$. If $\bar{f}_k = 0$, then in step 1.3 $\mathrm{Ext}(\widehat{F}_k) = \widehat{F}_k^{\mathcal{I}}$ is set to $\emptyset$, and the assertion is trivially satisfied. Otherwise, since $\bar{f}_k$ and all $\bar{a}_{kk'}$ satisfy inequalities 6.1 of $\Psi_{\mathcal{S}}$, it holds that

$$m \leq \left\lfloor \frac{\sum_{(k,k')\in I_P} \bar{a}_{kk'}}{\bar{f}_k} \right\rfloor \leq \left\lceil \frac{\sum_{(k,k')\in I_P} \bar{a}_{kk'}}{\bar{f}_k} \right\rceil .$$

On the other hand it is easy to see that for each element o_k^j of $\mathrm{Ext}(\widehat{F}_k)$, the number of pairs of $\mathrm{Ext}(A)$ that in step 2.2.2 get assigned o_k^j to the first component is

$$\text{either} \quad \left\lfloor \frac{\sum_{(k,k')\in I_P} \bar{a}_{kk'}}{\bar{f}_k} \right\rfloor \quad \text{or} \quad \left\lceil \frac{\sum_{(k,k')\in I_P} \bar{a}_{kk'}}{\bar{f}_k} \right\rceil ,$$

which shows that $\mathcal{I}$ satisfies the assertion. A similar argument shows that also assertions of the form $\widehat{F}_k \dot{\preceq} \exists^{\leq n} A$, or $\widehat{F}_k \dot{\preceq} \exists^{\geq m} A^-$, or $\widehat{F}_k \dot{\preceq} \exists^{\leq n} A^-$ in $\widehat{\mathcal{T}}$ are satisfied in $\mathcal{I}$.

"$\Rightarrow$" Notice that the construction of the model must guarantee that $A^{\mathcal{I}}$ is a set and not a multiset, i.e. that no two pairs of $\mathrm{Ext}(A)$ get assigned the same first and second components. Suppose that for some $(k, k') \in I_P$ condition 6.3 does not hold, i.e.

$$\bar{a}_{kk'} > \bar{f}_k \cdot \bar{f}_{k'}. \tag{6.4}$$

Assume there is a finite model $\mathcal{I}$ of $\widehat{\mathcal{S}}$ with $\bar{f}_k$ instances of $\widehat{F}_k$, $\bar{f}_{k'}$ instances of $\widehat{F}_{k'}$, and $\bar{a}_{kk'}$ different pairs in $A^{\mathcal{I}}$ whose first component is an instance of $\widehat{F}_k$ and whose second component is an instance of $\widehat{F}_{k'}$. Since the number of different pairs that can be formed from $\bar{f}_k$ different first components and $\bar{f}_{k'}$ different second components is $\bar{f}_k \cdot \bar{f}_{k'}$, if condition 6.4 holds, at least two pairs in $A^{\mathcal{I}}$ must be equal, contradicting the assumption that such a model $\mathcal{I}$ exists. □

The solutions of $\Psi_{\mathcal{S}}$ that correspond to finite models of $\mathcal{S}$ can now be characterized as those that are $\mathcal{W}_{\widehat{\mathcal{S}}}$-acceptable (according to Definition 6.1.3) with respect to a particular binary relation $\mathcal{W}_{\widehat{\mathcal{S}}}$. Let

$$\mathcal{W}_{\widehat{\mathcal{S}}} := \{(\mathrm{Var}(\widehat{P}), \mathrm{Var}(\widehat{F})) \mid \widehat{P} = A[\widehat{F}, \widehat{F}_2] \vee \widehat{P} = A[\widehat{F}_1, \widehat{F}]\}.$$

Definition 6.2.12 A solution *Sol* of $\Psi_{\mathcal{S}} := (\mathcal{V}, \mathcal{D})$ is said to be *acceptable* if it is $\mathcal{W}_{\widehat{\mathcal{S}}}$-acceptable (in the sense of Definition 6.1.3). ∎

Stated differently, an acceptable solution assigns the value 0 to an unknown $\mathrm{Var}(\widehat{P})$ corresponding to a compound attribute $\widehat{P} := A[\widehat{F}_1, \widehat{F}_2]$, if it assigns the value 0 either to $\mathrm{Var}(\widehat{F}_1)$ or to $\mathrm{Var}(\widehat{F}_2)$. Given a solution *Sol*, a finite model of the expansion is constructed by introducing for every compound class and every compound attribute a number of instances that is equal to the value assigned by *Sol* to the corresponding unknown. If this value is 0 for some compound class $\widehat{F}$, then $\widehat{\mathcal{F}}$ has an empty extension in the constructed model. Therefore every compound attribute that refers to this compound class must also have an empty extension, and *Sol* must assign the value 0 to the corrisponding unknown. The solution *Sol* satisfies this requirement only if it is acceptable.

In order to check whether a class C in a primitive $\mathcal{LUNI}$-schema is finitely consistent, we can proceed as follows: We construct the expansion of the schema an derive from it the system of inequalities. We then add an additional (non-homogenous) inequality that forces the solutions of the system to assign a positive value to at least one of the unknowns corresponding to the compound classes containing C. If this augmented system admits an acceptable solution, we can construct from such solution a finite model in which at least one of the compound classes containing C has a nonempty extension. This shows that C is consistent. On the contrary, if the schema admits a finite model in which C is populated, the number of instances of compound classes and attributes in such model provide directly a solution to the system of inequalities, and this solution is clearly acceptable.

This is formally proved in the following proposition, which represents the main result concerning finite consistency of a class in a primitive $\mathcal{LUNI}$-schema.

Proposition 6.2.13 *Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be a schema, $\widehat{\mathcal{S}} := (\mathcal{C}, \mathcal{A}, \widehat{\mathcal{F}}, \widehat{\mathcal{P}}, \widehat{\mathcal{T}})$ its expansion, $C \in \mathcal{C}$ a class name, and $\Psi_{\mathcal{S}} := (\mathcal{V}, \mathcal{D})$ the system of inequalities derived from $\mathcal{S}$. Then C is finitely consistent in $\mathcal{S}$ if and only if $\Psi^C_{\mathcal{S}} := (\mathcal{V}, \mathcal{D}^C)$, where*

$$\mathcal{D}^C := \mathcal{D} \bigcup \left\{ \sum_{\widehat{F} \in \widehat{\mathcal{F}} \mid C \in \widehat{F}} Var(\widehat{F}) \geq 1 \right\}$$

admits an acceptable nonnegative integer solution.

Proof. "⇐" Let *Sol* be an acceptable nonnegative integer solution. For each compound class and compound attribute $\widehat{X}$, let $Sol(\widehat{X})$ denote the value assigned by *Sol* to $\mathrm{Var}(\widehat{X})$.

We can assume that for every compound attribute $A[\widehat{F}_1, \widehat{F}_2] \in \widehat{\mathcal{P}}$ the following condition holds:

$$Sol(A[\widehat{F}_1, \widehat{F}_2]) \leq Sol(\widehat{F}_1) \cdot Sol(\widehat{F}_2). \tag{6.5}$$

In fact, since *Sol* is an acceptable solution, if $Sol(\widehat{F}_1) = 0$ or $Sol(\widehat{F}_2) = 0$, then $Sol(A[\widehat{F}_1, \widehat{F}_2]) = 0$ holds. Assume now that $Sol(\widehat{F}_1) \neq 0$ and $Sol(\widehat{F}_2) \neq 0$ and condition 6.5 does not hold for some compound attribute $A[\widehat{F}_1, \widehat{F}_2]$. Then it is sufficient to multiply *Sol* to a suitably large integer constant to obtain another integer solution *Sol′* of $\Psi^C_{\mathcal{S}}$ such that condition 6.5 is satisfied for $A[\widehat{F}_1, \widehat{F}_2]$. Since the only non-homogeneous inequality of $\Psi^C_{\mathcal{S}}$ is positive, part 1 of Lemma 6.1.2 guarantees that *Sol′* is indeed a solution of $\Psi^C_{\mathcal{S}}$.

For every attribute name $A \in \mathcal{A}$, let $\mathcal{S}_A := (\mathcal{C}, \mathcal{A}, \mathcal{T}_A)$ be the schema where $\mathcal{T}_A$ is obtained from $\mathcal{T}$ by removing all assertions involving an attribute different from A. Let $\widehat{\mathcal{S}}$ be the expansion of $\mathcal{S}$ and $\widehat{\mathcal{S}}_A$

the expansion of $\mathcal{S}_A$. Since $\mathcal{T}_A \subseteq \mathcal{T}$, each inequality of $\Psi_{\mathcal{S}_A}$ can be obtained from an inequality of $\Psi_{\mathcal{S}}$ by removing the unknowns corresponding to compound classes and compound attributes present in $\widehat{\mathcal{S}}_A$ but not present in $\widehat{\mathcal{S}}$. Therefore *Sol* is also a solution of $\Psi_{\mathcal{S}_A}$ if it is extended by assigning the value 0 to all additional unknowns in $\Psi_{\mathcal{S}_A}$.

Lemma 6.2.11 ensures that for every attribute $A \in \mathcal{A}$, a finite model $\mathcal{I}_A$ of $\widehat{\mathcal{S}}_A$ exists, with $Sol(\widehat{A})$ instances of $\widehat{A}$, for every $\mathcal{S}$-consistent compound attribute $\widehat{A}$ corresponding to A, and $Sol(\widehat{F})$ instances of $\widehat{F}$, for every $\mathcal{S}$-consistent compound class $\widehat{F}$ of $\widehat{\mathcal{S}}$. A finite model $\mathcal{I}$ of $\widehat{\mathcal{S}}$ can easily be obtained by merging the models $\mathcal{I}_A$ for all attributes A and unifying the sets of instances of each compound class. This can in fact be done since all models of the expansions $\widehat{\mathcal{S}}_A$ are constructed from the same solution of $\Psi^C_{\mathcal{S}}$ and therefore have the same number of instances for each compound class. By Lemma 6.2.8 $\mathcal{I}$ is also a model of $\mathcal{S}$. Moreover, since $\sharp\widehat{F}^{\mathcal{I}} = Sol(\widehat{F})$ for every compound class $\widehat{F}$, and since $\sum_{\widehat{F}|C\in\widehat{F}} Sol(\widehat{F}) \geq 1$, by applying Lemma 6.2.4 we have that

$$C^{\mathcal{I}} = \bigcup_{\widehat{F} \mid C\in\widehat{F}} \widehat{F}^{\mathcal{I}} \neq \emptyset.$$

We can conclude that C is finitely consistent in $\mathcal{S}$.

"$\Rightarrow$" Suppose a finite model $\mathcal{I}$ of $\mathcal{S}$ exists, where $C^{\mathcal{I}} \neq \emptyset$. Let *Sol* be the solution that assigns to each unknown $\mathrm{Var}(\widehat{X})$ the number of instances in $\mathcal{I}$ of the corresponding compound class or compound attribute $\widehat{X}$, i.e. $Sol(\widehat{X}) := \sharp\widehat{X}^{\mathcal{I}}$. *Sol* is obviously integer and nonnegative. Moreover it is acceptable, since if a compound attribute $A[\widehat{F}_1, \widehat{F}_2]$ has a nonempty extension, then both $\widehat{F}_1$ and $\widehat{F}_2$ must have a nonempty extension, and therefore both $Sol(\widehat{F}_1) \neq 0$ and $Sol(\widehat{F}_2) \neq 0$.

We show that *Sol* satisfies all inequalities in $\Psi^C_{\mathcal{S}}$. With respect to inequalities 6.1, let $(\widehat{F} \doteq \exists^{\geq m} A) \in \widehat{\mathcal{T}}$. Since $\mathcal{I}$ is a model of $\mathcal{S}$, by Lemma 6.2.8 it is also a model of $\widehat{\mathcal{S}}$ and therefore satisfies $\widehat{F} \doteq \exists^{\geq m} A$. It follows that for each instance o of $\widehat{F}$ in $\mathcal{I}$ there must be $k \geq m$ different pairs $(o, o_1), \ldots, (o, o_k) \in A^{\mathcal{I}}$. Therefore, the number of pairs $(o', o'') \in A^{\mathcal{I}}$ such that $o' \in \widehat{F}^{\mathcal{I}}$ must be at least m times the number of instances of $\widehat{F}$ in $\mathcal{I}$, which shows that the inequality 6.1 corresponding to the above assertion is satisfied. With respect to the inequalities corresponding to a compound assertion involving an inverse attribute and with respect to inequalities 6.2 we can proceed in a similar way. Finally, since $C^{\mathcal{I}} \neq \emptyset$, there must be at least one compound class $\widehat{F}$ such that $C \in \widehat{F}$ and $\widehat{F}^{\mathcal{I}} \neq \emptyset$. This shows that also the additional inequality in $\Psi^C_{\mathcal{S}}$ is satisfied. □

Since by Proposition 6.1.6 we can verify the existence of acceptable nonnegative integer solutions for a system of inequalities, we can make use of this result to effectively decide finite class consistency in primitive $\mathcal{LUNI}$- schemata.

Example 6.2.1 (cont.) In order to check whether the class `Root` is finitely consistent in $\mathcal{S}_0$, we add to the system $\Psi_{\mathcal{S}_n}$ the inequality that forces `R` or `RN` to be populated, obtaining

$$\begin{aligned}
&\mathtt{r} = \mathtt{o} = 0\\
&\mathtt{e}_{\mathtt{RN,O}} = \mathtt{e}_{\mathtt{ON,O}} = \mathtt{e}_{\mathtt{E,R}} = \mathtt{e}_{\mathtt{E,RN}} = 0\\
&\mathtt{rn} \;\leq\; \mathtt{e}_{\mathtt{RN,ON}} \;\leq\; 2{\cdot}\mathtt{rn}\\
&\mathtt{on} \;\leq\; \mathtt{e}_{\mathtt{ON,ON}} \;\leq\; 2{\cdot}\mathtt{on}\\
&\mathtt{on} \;=\; \mathtt{e}_{\mathtt{RN,ON}} + \mathtt{e}_{\mathtt{ON,ON}}\\
&\mathtt{r} + \mathtt{rn} \;\geq\; 1.
\end{aligned}$$

It is easy to see that the inequalities in $\Psi_{\mathcal{S}_n}$ force both `r` and `rn` to get assigned value 0, and therefore the whole system admits no solution. This shows that the class `Root` cannot be populated in any finite model of $\mathcal{S}_0$.

Similarly, to check whether the class `OtherNode` is finitely consistent in $\mathcal{S}_0$, we can add the following inequality to $\Psi_{\mathcal{S}_n}$:

$$\mathtt{o} + \mathtt{on} \;\geq\; 1.$$

The resulting system admits solutions which are all acceptable. By simplifying the system we can verify that every solution assigns 0 to all unknowns except to `on` and to $\mathtt{e}_{\mathtt{ON,ON}}$, to which it assigns the same value. Let *Sol* be such a solution which assigns to `on` and to $\mathtt{e}_{\mathtt{ON,ON}}$ the positive integer k. A finite model $\mathcal{I} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{S}_0$ can be obtained by applying the construction described in the proof of Lemma 6.2.11. Since `ON` must have k instances and is the only compound class with a nonempty extension, we define $\Delta^{\mathcal{I}} = \mathtt{ON}^{\mathcal{I}} = \mathtt{OtherNode}^{\mathcal{I}} :=$

$\{o_1, \ldots, o_k\}$. By construction we obtain then $(\texttt{edge}[\texttt{ON}, \texttt{ON}])^{\mathcal{I}} = \texttt{edge}^{\mathcal{I}} := \{(o_i, o_i) \mid i \in \{1, \ldots, k\}\}$, which indeed gives us a model of $\mathcal{S}_0$, although not one in which the `edge` relation has a tree-like structure. ∎

6.2.5 Upper Bounds for Finite Model Reasoning

Summing up the results of this section, and exploiting the results of Section 6.1 we obtain the following upper bound for finite class consistency.

Theorem 6.2.14 *Finite class consistency $\mathcal{S} \not\models_f E \equiv \bot$ in primitive $\mathcal{LUNI}$-schemata can be decided in worst case deterministic exponential time in $|\mathcal{S}| + |E|$.*

Proof. If E is not a class name, by Proposition 2.3.5, E is finitely consistent in $\mathcal{S}$ if and only if C is finitely consistent in $\mathcal{S}' := (\mathcal{C}', \mathcal{A}, \mathcal{T}')$, where $\mathcal{C}' := \mathcal{C} \cup \{C\}$ with $C \notin \mathcal{C}$ and $\mathcal{T}' := \mathcal{T} \cup \{C \doteq E\}$. Since $|\mathcal{S}'|$ is linear in $|\mathcal{S}| + |E|$, it is indeed sufficient to consider the case where E is a class name C.

By Lemma 6.2.2 we can construct a normalized schema $\mathcal{S}'$ equivalent to $\mathcal{S}$ such that $|\mathcal{S}'|$ is linear in $|\mathcal{S}|$. By Lemma 6.2.10 we can construct the corresponding system $\Psi^C_{S'}$ of linear inequalities in time at most exponential in $|\mathcal{S}|$ and therefore in $|\mathcal{S}'|$. By Proposition 6.2.13 C is finitely consistent in $\mathcal{S}'$, and by Lemma 2.3.7 in $\mathcal{S}$, if and only if $\Psi^C_{S'}$ admits an acceptable nonnegative integer solution. By Proposition 6.1.6 the existence of such solution can be verified in polynomial time in $|\Psi^C_{S'}|$, i.e. in worst case exponential time in $|\mathcal{S}|$. □

Corollary 6.2.15 *Finite class consistency in primitive $\mathcal{LUNI}$-schemata is **EXPTIME**-complete.*

Proof. The claim follows from Corollary 4.3.5 and 6.2.14. □

The method for deciding finite class consistency can also be adapted in order to decide finite class subsumption $\mathcal{S} \models_f E_1 \preceq E_2$ in relevant cases.

We discuss first the case where E_1 is a class name C and $E_2 = \forall L.E$, where E is a boolean combination of class names of $\mathcal{S}$, i.e. it is obtained by applying only the constructors "$\neg$", "$\sqcap$", and "$\sqcup$", starting from class names. In this case, it is not possible to directly use the algorithm for class consistency, but the method presented above can be extended as follows. By definition, $\mathcal{S} \not\models_f C \preceq \forall L.E$ if and only if there is a model $\mathcal{I}$ of $\mathcal{S}$ in which there is an instance o of C that is connected via link L to another object o' that is not an instance of E. Such condition can be directly verified by adding a suitable inequality to $\Psi_{\mathcal{S}}$. We show this in the case where L is an attribute A. The case where L is the inverse A^- of an attribute can be treated analogously.

We need a preliminary definition. Each compound class $\widehat{F}$, induces a truth assignment $\Phi_{\widehat{F}}$ on the class names in $\mathcal{C}$: the value assigned by $\Phi_{\widehat{F}}$ to a class name C is $\mathbf{t}$ if $C \in \widehat{F}$ and is $\mathbf{f}$ otherwise. This truth assignment can be extended in the obvious way to class expressions which are boolean combinations of class names. Given such a class expression E we say that $\widehat{F}$ *realizes* E, written as $E \dashv \widehat{F}$, if $\Phi_{\widehat{F}}(C) = \mathbf{t}$. Checking if $E \dashv \widehat{F}$ can be done in linear time with respect to the size of E. The following easy lemma shows how this syntactic definition has a semantic correspondence.

Lemma 6.2.16 *Let E be a boolean combination of class names in $\mathcal{C}$ and $\widehat{F}$ a compound class. Then for every interpretation $\mathcal{I}$ the following holds:*

$$\begin{aligned} E \dashv \widehat{F} &\implies E^{\mathcal{I}} \supseteq \widehat{F}^{\mathcal{I}} \\ E \not\dashv \widehat{F} &\implies E^{\mathcal{I}} \cap \widehat{F}^{\mathcal{I}} = \emptyset. \end{aligned}$$

Proof. The proof is an easy induction on the structure of E, using Lemma 6.2.4 for the base case. □

Proposition 6.2.17 *Let $\mathcal{S}$ be a schema, C a class name in $\mathcal{S}$, E a boolean combination of class names in $\mathcal{S}$, and $\Psi_{\mathcal{S}} := (\mathcal{V}, \mathcal{D})$ the system of inequalities derived from $\mathcal{S}$. Then $\mathcal{S} \models_f C \preceq \forall A.E$ if and only if the system $\Psi'_{\mathcal{S}} := (\mathcal{V}, \mathcal{D}')$, where*

$$\mathcal{D}' \;:=\; \mathcal{D} \bigcup \left\{ \sum_{\widehat{F} \in \widehat{\mathcal{F}} \mid C \in \widehat{F}} Var(\widehat{F}) \geq 1, \quad \sum_{\widehat{P} \in \mathcal{X}} Var(\widehat{P}) \geq 1 \right\},$$

and $\mathcal{X} := \{\widehat{P} \in \widehat{\mathcal{P}} \mid \exists \widehat{F}_1, \widehat{F}_2 \in \widehat{\mathcal{F}}\colon\ \widehat{P} = A[\widehat{F}_1, \widehat{F}_2] \wedge C \in \widehat{F}_1 \wedge {\sim}E \dashv \widehat{F}_2\}$*, does not admit an acceptable nonnegative integer solution.*

Proof. "$\Leftarrow$" Assume $\mathcal{S} \not\models_f C \preceq \forall A.E$. Then there is a model $\mathcal{I}$ of $\mathcal{S}$ in which there is an instance o of C that is connected via attribute A to another object o' that is not an instance of E. Let *Sol* be the solution that assigns to each unknown $\mathrm{Var}(\widehat{X})$ the number of instances in $\mathcal{I}$ of the corresponding compound class or compound attribute $\widehat{X}$. As shown in Proposition 6.2.13, since $\mathcal{I}$ is a model of $\mathcal{S}$ in which $C^{\mathcal{I}} \neq \emptyset$, *Sol* is an acceptable nonnegative integer solution that satisfies all inequalities in $\Psi_{\mathcal{S}}$, and also the additional inequality $\sum_{\widehat{F} \in \widehat{\mathcal{F}} \mid C \in \widehat{F}} \mathrm{Var}(\widehat{F}) \geq 1$.

Let $\widehat{F}$ be the unique compound class of which o is an instance, and $\widehat{F}'$ be the unique compound class of which o' is an instance. Let $\widehat{P} := A[\widehat{F}, \widehat{F}']$. From $(o, o') \in A^{\mathcal{I}}$, $o \in C^{\mathcal{I}}$, and $o' \in ({\sim}E)^{\mathcal{I}}$ it follows that $(o, o') \in \widehat{P}^{\mathcal{I}}$ and by Lemma 6.2.16 we have $C \in \widehat{F}$ and ${\sim}E \dashv \widehat{F}'$. Therefore $\widehat{P} \in \mathcal{X}$, and since it has a nonempty extension, $Sol(\widehat{P}) \geq 1$, and *Sol* satisfies also the inequality $\sum_{\widehat{P} \in \mathcal{X}} \mathrm{Var}(\widehat{P}) \geq 1$.

"$\Rightarrow$" Let *Sol* be an acceptable nonnegative integer solution of $\Psi'_{\mathcal{S}}$. By applying the construction in the proof of Lemma 6.2.11 we obtain a model $\mathcal{I}$ of $\mathcal{S}$ in which $C^{\mathcal{I}} \neq \emptyset$, and in which there is an instance (o, o') of a compound attribute $A[\widehat{F}, \widehat{F}']$, where $C \in \widehat{F}$ and ${\sim}E \dashv \widehat{F}'$. Therefore $o \in C^{\mathcal{I}}$, and $o' \notin E^{\mathcal{I}}$, from which we obtain $C^{\mathcal{I}} \not\subseteq (\forall A.E)^{\mathcal{I}}$. □

Theorem 6.2.18 *Finite class subsumption $\mathcal{S} \models_f E_1 \preceq E_2$ in primitive $\mathcal{LUNI}$-schemata can be decided in worst case deterministic exponential time in $|S| + |E_1| + |E_2|$, assuming E_2 has one of the following forms:*

1. *E_2 does not contain the constructor for universal quantification.*
2. *$E_2 = \forall L.E$, where E is a boolean combination of class names in $\mathcal{S}$.*

Proof. By Proposition 2.3.5 it is sufficient to consider the case where E_1 is a class name C.

In case (1), the negation normal form ${\sim}E_2$ of $\neg E_2$ (as specified in Definition 2.1.3) is an $\mathcal{LUNI}$-class expression. Therefore we can make use of Proposition 2.3.2 and reduce the problem of deciding $\mathcal{S} \models_f C \preceq E_2$ to the problem of deciding $\mathcal{S} \models_f C \sqcap {\sim}E_2 \equiv \bot$. By Theorem 6.2.14 this can be decided in worst case deterministic exponential time in $|\mathcal{S}| + |E_2|$.

In case (2), by Proposition 6.2.17 $\mathcal{S} \models_f C \preceq \forall L.E$ if and only if $\Psi'_{\mathcal{S}}$ does not admit an acceptable nonnegative integer solution. By Proposition 6.1.6 the existence of such solution can be verified in polynomial time in $|\Psi'_{\mathcal{S}}|$, i.e. in worst case exponential time in $|\mathcal{S}|$. □

We remark that the more general case, where also the subsumer is an arbitrary class expression, can be handled by the techniques introduced in Section 6.4, which however are more involved and of higher computational complexity.

6.3 Towards an Efficient Implementation

In this section we analyze in more detail the computational complexity of the method developed in Section 6.2 for verifying finite class consistency. In particular, we discuss how various assumptions on the structure of the schema that are commonly made in Databases may influence the complexity of the algorithm and lead to an efficient behaviour in relevant cases that are of practical interest.

The method introduced in Section 6.2 for verifying if a class C is finitely consistent in a primitive $\mathcal{LUNI}$-schema $\mathcal{S}_0$ suggests to split this task in three separate phases:

1. We normalize $\mathcal{S}_0$ and obtain an equivalent schema $\mathcal{S}$.
2. We construct the expansion $\widehat{\mathcal{S}}$ of $\mathcal{S}$.
3. We derive from $\widehat{\mathcal{S}}$ a system $\Psi^C_{\mathcal{S}}$ of linear inequalities and search for nonnegative acceptable integer solutions of $\Psi^C_{\mathcal{S}}$. If such a solution exists we can conclude that C is finitely consistent in $\mathcal{S}$ and therefore also in $\mathcal{S}_0$.

With respect to phase 3, we have seen that the construction of $\Psi_{\mathcal{S}}^{C}$ from $\widehat{\mathcal{S}}$ is polynomial, and that the problem of verifying the existence of a nonnegative acceptable integer solution of $\Psi_{\mathcal{S}}^{C}$ can be reduced to the problem of finding an arbitrary positive rational solution of a system of inequalities whose size is polynomial in $|\Psi_{\mathcal{S}}^{C}|$. Moreover, for the latter task we can rely on well studied and efficient linear programming methods. Therefore, we do not discuss phase 3 anymore and in the following we concentrate on the other two phases.

With respect to phase 1, we notice that the normalization of $\mathcal{S}_0$ requires the introduction of a number of new class names that is linear in size $\mathcal{S}_0$. Since the number of compound classes is exponential in the number of class names it is essential to keep this number as small as possible. Analyzing the reasoning procedure described in Section 6.2 we observe that it can be generalized to the case where the schema is not completely normalized but contains only assertions $C \doteq E$, where E is a class expression that has one of the following forms:

- E is a boolean combination of class names.
- $E = \forall L.E'$ where E' is a boolean combination of class names.
- $E = \exists^{\geq m} L$.
- $E = \exists^{\leq n} L$.

We call a schema that satisfies this condition *partially normalized*. The only modification of the presented reasoning method consists in a generalization of the definitions of consistent compound class and of consistent compound attribute in order to take into account the more general form of assertions in partially normalized schemata. All successive considerations apply then unmodified.

This observation allows us to reduce the number of new class names we have to introduce and therefore also the number of compound classes we have to consider for the construction of the expansion. In the rest of the section we assume to deal only with partially normalized schemata, and we concentrate on strategies for optimizing phase 2. We assume also that the boolean combinations of class names that appear in the schema are all in negation normal form.

6.3.1 Strategies for Constructing the Expansion

The proof of Lemma 6.2.6 suggests a trivial method to construct the expansion of a primitive (partially normalized) $\mathcal{LUNI}$-schema $\mathcal{S}$: Enumerate the exponentially many compound classes and compound attributes and check for each one in linear time if it is consistent. Proceeding in this way the construction of the expansion will necessarily require time exponential in $|\mathcal{S}|$. We propose now a strategy for improving the efficiency of this task in many relevant cases. Since the number of compound attributes is polynomially related to the number of compound classes, in the following we concentrate only on the issues related to determining the set of consistent compound classes.

If we analyze more in detail the factors that contribute to the complexity of this task, we can distinguish between different categories of schemata:

(α) Those schemata where the number of compound classes in the expansion is polynomial in the number of class names (or can be made polynomial without affecting the result of the class consistency checks).

(β) Those schemata where the number of compound classes in the expansion is exponential in the number of class names, but where we can *safely* assume that all but a polynomial number of them have an empty extension. Here, "safely" means that making this assumption does not influence consistency for any of the classes of the schema.

(γ) Those schemata where the number of compound classes in the expansion is necessarily exponential in the number of class names (where "necessarily" means that such a schema does not fall into category β).

If the schema we have to deal with falls into category (γ), any sound and complete procedure for verifying class consistency that is based on the construction of the expansion is deemed to work in exponential time. Therefore, the following considerations are devoted to describe a strategy that, when possible, does better than the trivial method of enumerating all compound classes and checking each one for consistency. This strategy may lead to an efficient treatment of the problem in those cases where the schema to be analyzed falls in category (α) or (β).

A preliminary analysis shows that in category (α) there must be schemata where, even if the size of the expansion is polynomial, nevertheless it takes exponential time to discard all inconsistent compound classes, and there must also be schemata where both the size of the expansion and the time to construct it can be kept polynomial.

Actually, we argue that in most practical situations, this last case is the most likely to occur. In the following we propose a heuristics for optimizing the selection of consistent compound classes, which performs this process in two steps: In the first step, a *pre-selection* on the compound classes is performed, using an efficient and possibly incomplete algorithm that allows us to discard a priori as many of them as possible. In the second step each of the remaining compound classes is then checked for consistency, and only the consistent ones are taken into account. Since this test can be performed in linear time for each compound class, the complexity of the second step essentially depends on the number of compound classes that remain to be considered after the pre-selection.

The pre-selection is done by considering each pair of classes in turn, and trying to extract from the schema as much information as possible concerning both inclusion and disjointness for the classes in each pair. This information is used on one hand for discarding inconsistent compound classes, and on the other hand for singling out those consistent compound classes that can be ignored without influencing the correctness of consistency checking. Of course there is a tradeoff between the need to be efficient in the pre-selection and the need to obtain from it as much information as possible.

We propose to construct two data structures, one, called *disjointness table*, for storing pairs of classes that are disjoint in every model of the schema, and one, called *inclusion table*, for storing pairs of classes such that the first class of the pair is necessarily included in the second. Basically, the pre-selection step consists in filling in the entries of the two tables. We have determined two criteria that we can follow to do this:

(a) To consider inclusion or disjointness that logically follows from the propositional part of the assertions in the schema.

(b) To consider inclusion or disjointness that may be assumed without influencing consistency checking for any class in the schema.

Regarding criterion (a), the simplest strategy is to fill the entries of the tables simply considering inclusion and disjointness that are explicitly present in the schema. If, for example, the schema contains an assertion $C_1 \doteq \neg C_2$, this allows us immediately to exclude all compound classes containing both C_1 and C_2. A more sophisticated method is to consider inclusion or disjointness of two classes that are deducible from the whole propositional part of the schema. However, if we pose no restrictions on the assertions that are taken into account, the problem of determining when inclusion or disjointness between classes logically follows from the propositional part of the schema, is **coNP**-complete, since it corresponds to propositional validity. Nevertheless, since we are performing a pre-selection, it may be sufficient to use an efficient and sound procedure that does not guarantee completeness (see for example [51]).

With respect to criterion (b), we propose a simple syntactic method that can be applied in order to introduce disjointness between classes without influencing satisfiability of classes in the schema. Let $\mathcal{G}_\mathcal{S} := (V, E)$ be the undirected graph constructed from a (partially normalized) primitive $\mathcal{LUNI}$-schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ in the following way:

1. For each class $C \in \mathcal{C}$, introduce one node n_C in V;
2. For each pair of classes $C_1, C_2 \in \mathcal{C}$, introduce in E an edge between n_{C_1} and n_{C_2} if at least one of the following conditions is satisfied:
   - $\mathcal{T}$ contains an assertion $C_1 \doteq E$, where E is a boolean combination of class names of $\mathcal{C}$ in which C_2 appears positively[3] (i.e. not preceded by the symbol "$\neg$").
   - The schema contains an assertion $C \doteq E$, or an assertion $C \doteq \forall L.E$, where $C \in \mathcal{C}$ and E is a boolean combination of class names of $\mathcal{C}$ in which both C_1 and C_2 appear positively.
3. For each pair of classes $C_1, C_2 \in \mathcal{C}$, remove (if present) the edge between n_{C_1} and n_{C_2} if C_1 and C_2 have been determined to be disjoint by applying criterion (a) above.

[3] We remind that we assume that E is in negation normal form.

The following theorem, based on the construction of such a graph, allows us to augment the number of entries in the disjointness table, thus reducing the number of compound classes that actually have to be taken into account for the construction of the expansion.

Theorem 6.3.1 *Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be a (partially normalized) primitive $\mathcal{LUNI}$-schema and let $\mathcal{G}_\mathcal{S}$ be the graph derived from $\mathcal{S}$ as stated above. Let $\mathcal{T}'$ be the set of assertions obtained from $\mathcal{T}$ by adding an assertion $C_1 \dot{\preceq} \neg C_2$ for each pair of classes $C_1, C_2 \in \mathcal{C}$ such that n_{C_1} and n_{C_2} are not connected in $\mathcal{G}_\mathcal{S}$. Then for each class $C \in \mathcal{C}$ we have that C is finitely consistent in $\mathcal{S}$ if and only if it is finitely consistent in $\mathcal{S}' := (\mathcal{C}, \mathcal{A}, \mathcal{T}')$.*

Proof (sketch). Let $C_1, C_2 \in \mathcal{C}$ be such that n_{C_1} and n_{C_2} are not connected in $\mathcal{G}_\mathcal{S}$, and let $\mathcal{I}$ be a model of $\mathcal{S}$ such that there is an $o \in \Delta^\mathcal{I}$ with $o \in C_1^\mathcal{I} \cap C_2^\mathcal{I}$. Then we can construct an interpretation $\mathcal{I}'$ of $\mathcal{S}$ as follows:

- $\Delta^{\mathcal{I}'} := \Delta^\mathcal{I} \cup \{o_1, o_2\} \setminus \{o\}$.
- $o_1 \in C_1^{\mathcal{I}'} \setminus C_2^{\mathcal{I}'}$ and $o_2 \in C_2^{\mathcal{I}'} \setminus C_1^{\mathcal{I}'}$.
- For any class $C \in \mathcal{C}$ such that $o \notin C^\mathcal{I}$, we put $C^{\mathcal{I}'} := C^\mathcal{I}$, and for any class $C \in \mathcal{C}$ such that $o \in C^\mathcal{I}$, we put either $C^{\mathcal{I}'} := C^\mathcal{I} \cup \{o_1\} \setminus \{o\}$ or $C^{\mathcal{I}'} := C^\mathcal{I} \cup \{o_2\} \setminus \{o\}$.
- For any attribute in $A \in \mathcal{A}$ we substitute each pair in which one of the components is o either with a pair in which the same component is o_1 or with one in which it is o_2.

It is possible to show that since n_{C_1} and n_{C_2} are not connected in $\mathcal{G}_\mathcal{S}$, we can make the choices in the construction of $\mathcal{I}'$ in such a way that $\mathcal{I}'$ is a model of $\mathcal{S}$. Proceeding by induction on the number of objects in $\Delta^\mathcal{I}$ we can effectively construct a finite model of $\mathcal{S}$ in which the required condition is satisfied. □

Once we have constructed the disjointness and inclusion tables, the information therein is used to cut down the number of compound classes that have to be checked for consistency. Each entry in the tables allows us to exclude all compound classes that do not satisfy the inclusion or disjointness condition specified by the entry, and therefore excludes three quarters of the total number of compound classes.

6.3.2 Special Cases

We discuss now some meaningful cases in which the proposed method for finite class consistency may work in polynomial time in the size of the schema.

One case that is worth mentioning is when we actually deal with a primitive $\mathcal{LNI}$-schema, i.e. the class expressions in the schema do not contain the constructor for disjunction. In such case the construction of the inclusion and disjointness tables represents an optimal strategy in the following sense: On one hand all possible consequences of the assertions that do not involve number restrictions and universal quantification can be computed in polynomial time [7], and on the other hand, by applying the method referred to in Theorem 6.3.1, we can complete the disjointness table in such a way that the number of disjointness assertions is maximized. It is possible to show that in this case all inconsistent compound classes are determined by using the information contained in the two tables, and, moreover, all possible disjointness assumptions between classes are derived, in the sense that introducing further ones may influence the correctness of class consistency checking.

A meaningful case where the number of compound classes can be reduced dramatically, is when the disjointness assertions induce a partition of the classes into a number of clusters, such that classes belonging to different clusters are disjoint. The clusters can easily be determined by constructing a graph with one node for each class and an edge between two nodes if and only if the corresponding classes are not disjoint. Each cluster corresponds to a connected component of this graph. In such a case, all compound classes we have to consider are formed only of classes of the same cluster, and the set of compound classes becomes the union of the sets of compound classes obtained separately for each cluster. If we can ensure that for each cluster this number is polynomial, we obtain a system of inequalities whose size is polynomial in the size of the original schema. This is the case, for example, if the size of each cluster is logarithmic in the total number of classes.

Another special case to be considered is that of $\mathcal{LNI}$-schemata where the classes are organized in *generalization hierarchies*. Generalization hierarchies are treelike structures representing inclusion, where it is

Constructor Name	Syntax	Semantics
class name	C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
general negation ($\mathcal{C}$)	$\neg E$	$\Delta^{\mathcal{I}} \setminus E^{\mathcal{I}}$
conjunction	$E_1 \sqcap E_2$	$E_1^{\mathcal{I}} \cap E_2^{\mathcal{I}}$
disjunction	$E_1 \sqcup E_2$	$E_1^{\mathcal{I}} \cup E_2^{\mathcal{I}}$
universal quantification	$\forall A.E$	$\{o \mid \forall o' \colon (o,o') \in L^{\mathcal{I}} \rightarrow o' \in E^{\mathcal{I}}\}$
existential quantification	$\exists A.E$	$\{o \mid \exists o' \colon (o,o') \in L^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}}\}$
number restrictions ($\mathcal{N}$)	$\exists^{\geq m} L$	$\{o \mid \sharp\{o' \mid (o,o') \in L^{\mathcal{I}}\} \geq m\}$
	$\exists^{\leq n} L$	$\{o \mid \sharp\{o' \mid (o,o') \in L^{\mathcal{I}}\} \leq n\}$
attribute name	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
inverse attribute ($\mathcal{I}$)	A^-	$\{(o,o') \mid (o',o) \in A^{\mathcal{I}}\}$
free assertion	$E_1 \dot{\preceq} E_2$	$E_1^{\mathcal{I}} \subseteq E_2^{\mathcal{I}}$

Table 6.1: Free $\mathcal{LCNI}$-schemata

assumed that classes in different groups are pairwise disjoint, and within one group the same holds for all classes at the same depth in the tree [20]. In this case, each group corresponds to one cluster, and for each cluster the number of compound classes equals the number of classes, since it corresponds to the number of paths from the root of the tree to a class. It follows that our method, when applied to schemas of this type, works in polynomial time. This is particularly important, if one considers that most object-oriented data models assume, either implicitly or explicitly, an organization of classes based on generalization hierarchies (see for example how isa-relationships are treated in [2]).

Finally, an important case where the proposed method works in polynomial time, is if one makes the *most specific class assumption*. Such assumption, which is common in many object-oriented data models, requires that if in a model of a schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ an object belongs to two different classes $C_1, C_2 \in \mathcal{C}$, then there is another class $C \in \mathcal{C}$ such that $\mathcal{T}$ contains the assertions $C \dot{\preceq} C_1$ and $C \dot{\preceq} C_2$. Under the most specific class assumption, for every compound class that may be populated in a model of the schema there must be a corresponding class name in $\mathcal{C}$. Therefore, the number of compound classes is not greater than the number of class names, and the size of the expansion is polynomial in $|\mathcal{S}|$.

6.4 Finite Model Reasoning on Free $\mathcal{LCNI}$-Schemata

In this section we extend the method introduced in Section 6.2 to solve the problem of reasoning on free $\mathcal{LCNI}$-Schemata with respect to finite models. Table 6.1 recalls the types of constructors and assertions that are allowed in free $\mathcal{LCNI}$-schemata.

We construct again a system of linear inequalities and relate the existence of models for the schema to the existence of particular solutions of the system. However the presence of arbitrary free inclusion assertions, and the higher expressivity of the underlying language (in particular the presence of qualified existential quantification) make the construction of the system much more involved than in the previous case. Indeed, while for primitive $\mathcal{LUNI}$-schemata it is sufficient to construct a system of inequalities whose size is simply exponential in the size of the schema, the technique presented in this section requires an additional exponential blowup, so that the size of the resulting system is doubly exponential in the size of the schema.

6.4.1 Normalization of a Schema

In the following we assume that all class expressions that appear in the schema are in negation normal form. This explains why in Table 6.1 we have chosen a set of constructors that is not minimal. We consider again a preliminary transformation of the schema in a normalized form. An $\mathcal{LCNI}$-class expression is called *normalized* if it is of the form[4]:

$$D \mid D_1 \sqcap D_2 \mid D_1 \sqcup D_2 \mid \forall L.D \mid \exists L.D \mid \exists^{\geq m} L \mid \exists^{\leq n} L.$$

[4] we remind that the letter D ranges over class literals

An assertion is said to be *normalized*, if the class expression on the left hand side is a class literal and the expression on the right hand side is normalized. A free $\mathcal{LCNI}$-schema is *normalized* if it is constituted solely by normalized assertions.

Lemma 6.4.1 *Every free $\mathcal{LCNI}$-schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ can be transformed in linear time into an equivalent normalized $\mathcal{LCNI}$-schema $\mathcal{S}' := (\mathcal{C}', \mathcal{A}, \mathcal{T}')$, where $\mathcal{C} \subseteq \mathcal{C}'$.*

Proof. In the course of the transformation we introduce also assertions of the form $D \doteq E$, which will be eliminated in the last step. The symbol "$\dot{\propto}$" denotes either "$\dot{\preceq}$" or "$\doteq$". The normalized schema $\mathcal{S}'$ equivalent to $\mathcal{S}$ can be obtained through the following procedure:

1. For each assertion $E \dot{\preceq} E'$ in $\mathcal{T}$, if E is not a literal then introduce a new class name C, and replace $E \dot{\preceq} E'$ with $C \doteq E$, and $C \dot{\preceq} E'$. This results in a schema $\mathcal{S}_1 = (\mathcal{C}_1, \mathcal{A}, \mathcal{T}_1)$ where $\mathcal{C}_1 \supseteq \mathcal{C}$ and all assertions in $\mathcal{T}_1$ have a literal on their left hand side.

2. Repeat the following, until all assertions in $\mathcal{T}_1$ are normalized: For each assertion $D \dot{\propto} E$ in $\mathcal{T}_1$, if E is not normalized then remove $D \dot{\propto} E$ from $\mathcal{T}_1$ and do the following:

   - If $E = E_1 \sqcap E_2$, then add two new class names C_1 and C_2 to $\mathcal{C}_1$ and add $D \dot{\propto} C_1 \sqcap C_2$, $C_1 \doteq E_1$, and $C_2 \doteq E_2$ to $\mathcal{T}_1$.
   - If $E = E_1 \sqcup E_2$, then add two new class names C_1 and C_2 to $\mathcal{C}_1$ and add $D \dot{\propto} C_1 \sqcup C_2$, $C_1 \doteq E_1$, and $C_2 \doteq E_2$ to $\mathcal{T}_1$.
   - If $E = \forall L.E_1$, then add a new class name C to $\mathcal{C}_1$ and add $D \dot{\propto} \forall L.C$ and $C \doteq E_1$ to $\mathcal{T}_1$.
   - If $E = \exists L.E_1$, then add a new class name C to $\mathcal{C}_1$ and add $D \dot{\propto} \exists L.C$ and $C \doteq E_1$ to $\mathcal{T}_1$.

   This results in a schema $\mathcal{S}_2 = (\mathcal{C}_2, \mathcal{A}, \mathcal{T}_2)$ where $\mathcal{C}_2 \supseteq \mathcal{C}$ and $\mathcal{T}_2$ contains only normalized assertions.

3. Replace each assertion $D \doteq E$ in $\mathcal{T}_2$ with the pair of assertions $D \dot{\preceq} E$ and $\sim D \dot{\preceq} \sim E$. This results in the normalized schema $\mathcal{S}'$.

Step (1) of the above construction introduces at most one new class name for each assertion in $\mathcal{T}$, and $|\mathcal{S}_1|$ is linear in $|\mathcal{S}|$. Step (2) introduces at most one new class name for each subexpression of a class expression in $\mathcal{T}_1$. Since the number of such subexpressions is linear in $|\mathcal{T}_1|$, at most a linear number of new classes is introduced, and $|\mathcal{S}_2|$ is linear in $|\mathcal{S}_1|$. It is also easy to see that the construction can be performed in linear time in $|\mathcal{S}_1|$. Step (2) does not introduce any new classes and only assertions of the form $D \doteq E$ in $\mathcal{T}_2$ are replaced by two assertions of at most double size. Therefore $|\mathcal{S}'|$ is linear in $|\mathcal{S}|$. It is also easy to see that all steps of the construction can be performed in linear time in $|\mathcal{S}|$. By construction $\mathcal{S}'$ is normalized and $\mathcal{C} \subseteq \mathcal{C}'$.

It remains to show that $\mathcal{S}$ and $\mathcal{S}'$ are equivalent. The equivalence of $\mathcal{S}$ and $\mathcal{S}_1$ and of $\mathcal{S}_2$ and $\mathcal{S}'$ follows immediately from the set-theoretic semantics. The equivalence of $\mathcal{S}_1$ and $\mathcal{S}_2$ can be shown by induction on the number of sub-steps of step (2), as in the proof of Lemma 6.2.2 □

Lemmata 6.4.1 and 2.3.7 allow us to restrict our attention to normalized schemata when devising procedures to perform the reasoning services. Moreover, if we replace each assertion of the form $D \dot{\preceq} D_1 \sqcap D_2$ with the pair of assertions $D \dot{\preceq} D_1$ and $D \dot{\preceq} D_2$, we obtain a schema that is equivalent to the original one and of linear size. Therefore, in the rest of the section we assume that the schemata we deal with are all free normalized $\mathcal{LCNI}$-schemata in which the "$\sqcap$" operator does not appear. Where not specified otherwise, we assume also that $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$.

6.4.2 Expansion of a Schema

We generalize now the definition of expansion of a schema introduced in Section 6.2.2 to free $\mathcal{LCNI}$-schemata. Differently from the previous case the expansion represents only an intermediate representation and it cannot be used directly to derive a system of linear inequalities. The notions of compound class and compound attribute remain unchanged, while compound assertions are generalized.

Definition 6.4.2 An *expanded schema* is a tuple $\widehat{\mathcal{S}} = (\mathcal{C}, \mathcal{A}, \widehat{\mathcal{F}}, \widehat{\mathcal{P}}, \widehat{\mathcal{T}})$, where

- $\mathcal{C}$ is a set of class names.
- $\mathcal{A}$ is a set of attribute names.
- $\widehat{\mathcal{F}} \subseteq 2^{\mathcal{C}}$ is a set of compound classes.
- $\widehat{\mathcal{P}} \subseteq \mathcal{A}\times\widehat{\mathcal{F}}\times\widehat{\mathcal{F}}$ is a set of compound attributes.
- $\widehat{\mathcal{T}}$ is a set of *compound assertions*. Each such assertion has one of the forms

$$\begin{array}{lcll} \widehat{F} & \dot{\preceq} & \exists^{\geq m} L, & \text{or} \\ \widehat{F} & \dot{\preceq} & \exists^{\leq n} L, & \text{or} \\ \widehat{F} & \dot{\preceq} & \exists L.D, & \end{array}$$

  where $\widehat{F} \in \widehat{\mathcal{F}}$, L is a link, i.e. either an attribute name in $\mathcal{A}$ or the inverse of an attribute name, D is a literal, i.e. either a class name in $\mathcal{C}$ or its negation, m is a positive, and n a nonnegative integer. ∎

Where not specified otherwise, we assume $\widehat{\mathcal{S}} = (\mathcal{C}, \mathcal{A}, \widehat{\mathcal{F}}, \widehat{\mathcal{P}}, \widehat{\mathcal{T}})$.

The semantics of compound classes, compound attributes and the notion of model of an expanded schema are defined exactly as in Section 6.2.2, considering that satisfaction of a compound assertion is defined also for assertions of the form $\widehat{F} \dot{\preceq} \exists L.D$. Generalizing Definition 6.2.5 we associate to each schema an expanded schema.

Definition 6.4.3 An expanded schema $\widehat{\mathcal{S}} := (\mathcal{C}, \mathcal{A}, \widehat{\mathcal{F}}, \widehat{\mathcal{P}}, \widehat{\mathcal{T}})$ is called the *expansion* of a schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ if the following conditions hold:

- $\widehat{\mathcal{F}}$ is the set of all $\mathcal{S}$-*consistent* compound classes, where a compound class $\widehat{F} \in 2^{\mathcal{C}}$ is said to be $\mathcal{S}$-consistent, if
  - for every $D \in \widehat{F}$, if $(D \dot{\preceq} D') \in \mathcal{T}$, then $D' \in \widehat{F}$.
  - for every $D \in \widehat{F}$, if $(D \dot{\preceq} D_1 \sqcup D_2) \in \mathcal{T}$, then $D_1 \in \widehat{F}$ or $D_2 \in \widehat{F}$.
- $\widehat{\mathcal{P}}$ is the set of all $\mathcal{S}$-*consistent* compound attributes, where a compound attribute $A[\widehat{F}_1, \widehat{F}_2] \in \mathcal{A}\times 2^{\mathcal{C}}\times 2^{\mathcal{C}}$ is said to be $\mathcal{S}$-consistent, if
  - $\widehat{F}_1$ and $\widehat{F}_2$ are $\mathcal{S}$-consistent.
  - for every $D \in \widehat{F}_1$, if $(D \dot{\preceq} \forall A.D') \in \mathcal{T}$, then $D' \in \widehat{F}_2$.
  - for every $D \in \widehat{F}_2$, if $(D \dot{\preceq} \forall A^{-1}.D') \in \mathcal{T}$, then $D' \in \widehat{F}_1$.
- $\widehat{\mathcal{T}}$ is the smallest set of compound assertions such that for every $\widehat{F} \in \widehat{\mathcal{F}}$
  - if for some $D \in \widehat{F}$ there is an assertion $(D \dot{\preceq} \exists^{\geq m} L) \in \mathcal{T}$, then $\widehat{\mathcal{T}}$ contains the compound assertion $\widehat{F} \dot{\preceq} \exists^{\geq m_{max}} L$, where
$$m_{max} := \max\{m \mid \exists D \in \widehat{F} \colon (D \dot{\preceq} \exists^{\geq m} L) \in \mathcal{T}\}.$$
  - if for some $D \in \widehat{F}$ there is an assertion $(D \dot{\preceq} \exists^{\leq n} L) \in \mathcal{T}$, then $\widehat{\mathcal{T}}$ contains the compound assertion $\widehat{F} \dot{\preceq} \exists^{\leq n_{min}} L$, where
$$n_{min} := \min\{n \mid \exists D \in \widehat{F} \colon (D \dot{\preceq} \exists^{\leq n} L) \in \mathcal{T}\}.$$
  - if for some $D \in \widehat{F}$ there is an assertion $(D \dot{\preceq} \exists L.D') \in \mathcal{T}$, then $\widehat{\mathcal{T}}$ contains the compound assertion $\widehat{F} \dot{\preceq} \exists L.D'$. ∎

It is easy to see that in general the size of the expansion is exponential in the size of the original schema. The following lemma shows that it can also be effectively constructed in exponential time.

Lemma 6.4.4 *The expansion $\widehat{\mathcal{S}}$ of $\mathcal{S}$ can be constructed in time which is exponential in $|\mathcal{S}|$.*

Proof. Analogous to the proof of Lemma 6.2.6. □

The following lemmata relate the (finite) models of a schema to the (finite) models of its expansion. Their proof easily extends the proofs of Lemmata 6.2.8 and 6.2.7, but we repeat it here for completeness.

Lemma 6.4.5 *Let $\mathcal{S}$ be a schema and $\mathcal{I}$ a (finite) model of $\mathcal{S}$. Then for each $\mathcal{S}$-inconsistent compound class or compound attribute X, $X^{\mathcal{I}} = \emptyset$.*

Proof. Let $\widehat{F} \in 2^{\mathcal{C}}$ be $\mathcal{S}$-inconsistent. Then (a) there is an assertion $(D \dot{\preceq} D') \in \mathcal{T}$ such that $D \in \widehat{F}$ and $D' \notin \widehat{F}$, or (b) there is an assertion $(D \dot{\preceq} D_1 \sqcup D_2) \in \mathcal{T}$ such that $D \in \widehat{F}$, $D_1 \notin \widehat{F}$, and $D_2 \notin \widehat{F}$. In case (a) $\widehat{F}^{\mathcal{I}} \subseteq D^{\mathcal{I}} \setminus D'^{\mathcal{I}}$, and since $\mathcal{I}$ satisfies $D \dot{\preceq} D'$, $\widehat{F}^{\mathcal{I}} = \emptyset$. In case (b) $\widehat{F}^{\mathcal{I}} \subseteq D^{\mathcal{I}} \setminus (D_1^{\mathcal{I}} \cup D_2^{\mathcal{I}})$, and since $\mathcal{I}$ satisfies $D \dot{\preceq} D_1 \sqcup D_2$, $\widehat{F}^{\mathcal{I}} = \emptyset$.

Let $\widehat{P} = A[\widehat{F}_1, \widehat{F}_2] \in \mathcal{A} \times 2^{\mathcal{C}} \times 2^{\mathcal{C}}$ be $\mathcal{S}$-inconsistent. Then (a) $\widehat{F}_1$ or $\widehat{F}_2$ is $\mathcal{S}$-inconsistent, or (b) there is an assertion $(D \dot{\preceq} \forall A.D') \in \mathcal{T}$ such that $D \in \widehat{F}_1$ and $D' \notin \widehat{F}_2$, or (c) there is an assertion $(CD \dot{\preceq} \forall A^{-}.D') \in \mathcal{T}$ such that $D \in \widehat{F}_2$ and $D' \notin \widehat{F}_1$. In case (a) $\widehat{F}_1^{\mathcal{I}} = \emptyset$ or $\widehat{F}_2^{\mathcal{I}} = \emptyset$, and therefore $\widehat{P}^{\mathcal{I}} \subseteq \widehat{F}_1^{\mathcal{I}} \times \widehat{F}_2^{\mathcal{I}} = \emptyset$. In case (b), assume by contradiction that $(o_1, o_2) \in \widehat{P}^{\mathcal{I}} \subseteq A^{\mathcal{I}}$ for some $o_1, o_2 \in \Delta^{\mathcal{I}}$. Then $o_1 \in \widehat{F}_1^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and $o_2 \in \widehat{F}_2^{\mathcal{I}}$. Since $\mathcal{I}$ satisfies $D \dot{\preceq} \forall A.D'$, it follows that $o_2 \in D'^{\mathcal{I}}$. But $\widehat{F}_2^{\mathcal{I}} \cap D'^{\mathcal{I}} = \emptyset$ gives rise to a contradiction. Case (c) can be treated in a similar way. □

Lemma 6.4.6 *Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be a schema, $\widehat{\mathcal{S}}$ its expansion, and $\mathcal{I}$ a (finite) interpretation over $\mathcal{C}$ and $\mathcal{A}$. Then $\mathcal{I}$ is a (finite) model of $\mathcal{S}$ if and only if it is a (finite) model of $\widehat{\mathcal{S}}$.*

Proof. "⇒" Let $\mathcal{I}$ be a (finite) model of $\mathcal{S}$. Since $\widehat{\mathcal{F}}$ contains exactly the set of $\mathcal{S}$-consistent compound classes, by Lemma 6.4.5, for a compound class $\widehat{F} \in 2^{\mathcal{C}} \setminus \widehat{\mathcal{F}}$ we have that $\widehat{F}^{\mathcal{I}} = \emptyset$. The same argument holds for compound attributes. Let $(\widehat{F} \dot{\preceq} \exists^{\geq m} L) \in \widehat{\mathcal{T}}$. Then there is a $D \in \widehat{F}$ such that $(D \dot{\preceq} \exists^{\geq m} L) \in \mathcal{T}$. Since $\mathcal{I}$ satisfies this assertion and $\widehat{F}^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, $\mathcal{I}$ satisfies also $\widehat{F} \dot{\preceq} \exists^{\geq m} L$. A similar argument holds if $(\widehat{F} \dot{\preceq} \exists^{\leq n} L) \in \widehat{\mathcal{T}}$. or $(\widehat{F} \dot{\preceq} \exists L.D') \in \widehat{\mathcal{T}}$.

"⇐" Let $\mathcal{I}$ be a (finite) model of $\widehat{\mathcal{S}}$ and assume by contradiction that there is an assertion $(D \dot{\preceq} E) \in \mathcal{T}$ not satisfied by $\mathcal{I}$. This means there is some $o \in \Delta^{\mathcal{I}}$ such that $o \in D^{\mathcal{I}}$ and $o \notin E^{\mathcal{I}}$. Let $\widehat{F}_o$ be the (unique) compound class such that $o \in \widehat{F}_o^{\mathcal{I}}$. Then we have that $D \in \widehat{F}_o$. Suppose $E = D'$. Since $o \notin D'^{\mathcal{I}}$, we have that $D' \notin \widehat{F}_o$. It follows that $\widehat{F}_o$ is not $\mathcal{S}$-consistent, which by Lemma 6.4.5 contradicts $o \in \widehat{F}_o^{\mathcal{I}}$. The case where $E = D_1 \sqcup D_2$ can be handled in a similar way. Suppose $E = \forall L.D'$. Since $o \notin (\forall A.D')^{\mathcal{I}}$ there is some $o' \in \Delta^{\mathcal{I}}$ such that $(o, o') \in A^{\mathcal{I}}$ and $o' \notin D'^{\mathcal{I}}$. Let $\widehat{F}_{o'}$ be the (unique) compound class such that $o' \in \widehat{F}_{o'}^{\mathcal{I}}$. Then $D' \notin \widehat{F}_{o'}$ and therefore $A[\widehat{F}_o, \widehat{F}_{o'}]$ is not consistent. By Lemma 6.4.5 this contradicts $(o, o') \in A^{\mathcal{I}}$, $a \in \widehat{F}_o^{\mathcal{I}}$, and $o' \in \widehat{F}_{o'}^{\mathcal{I}}$. The case where $E = \forall A^{-}.D'$ can be handled in a similar way. Suppose $E = \exists^{\geq m} L$. Since $D \in \widehat{F}_o$, $\widehat{\mathcal{T}}$ contains an assertion $\widehat{F}_o \dot{\preceq} \exists^{\geq m_{max}} L$, where $m_{max} \geq m$. From $o \notin (\exists^{\geq m} L)^{\mathcal{I}}$ it follows that $o \notin (\exists^{\geq m_{max}} L)^{\mathcal{I}}$, which contradicts the fact that $\mathcal{I}$ satisfies all assertions in $\widehat{\mathcal{T}}$. The case where $E = \exists^{\leq n} L$ can be handled in a similar way. Suppose $E = \exists L.D'$. Since $D \in \widehat{F}_o$, $\widehat{\mathcal{T}}$ contains the assertion $\widehat{F}_o \dot{\preceq} \exists L.D'$. The assumption $o \notin (\exists L.D')^{\mathcal{I}}$ contradicts the fact that $\mathcal{I}$ satisfies all assertions in $\widehat{\mathcal{T}}$. □

In Section 6.2 we have shown that for primitive $\mathcal{LUNI}$-schemata a system of linear inequalities can be directly constructed from the expansion, such that the acceptable nonnegative integer solutions of this system correspond to finite models of the schema. Unfortunately, for free $\mathcal{LCNI}$-schemata this approach does not work directly, and this is due to the presence of assertions of the form $C \dot{\preceq} \exists L.D$. One could think to handle such assertions by constructing the expansion as specified above, simply coding existential quantifications inside compound assertions and leaving their treatment to the system of inequalities (as done for number restrictions). The most natural extension of the system of inequalities would be to add for each assertion of the form $D_1 \dot{\preceq} \exists A.D_2$ an inequality

$$\sum_{\substack{\widehat{A[F_1, F_2]} \mid \\ D_1 \in \widehat{F_1} \wedge D_2 \in \widehat{F_2}}} \mathrm{Var}(A[\widehat{F}_1, \widehat{F}_2]) \;\geq\; \sum_{\widehat{F} \mid D_1 \in \widehat{F}} \mathrm{Var}(\widehat{F}),$$

and similarly for each assertion of the form $D_1 \dot{\preceq} \exists A^{-}.D_2$. One could think that imposing such conditions would guarantee, that from each acceptable nonnegative integer solution of the system a model of the schema

could be constructed, in which the number of instances of each compound class and compound attribute is given by the value assigned by the solution to the corresponding unknown. The following example shows that this is not the case in general.

Example 6.4.7 Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be the $\mathcal{LCNI}$-schema where

$\mathcal{C}_b := \{\mathtt{C}, \mathtt{D}_1, \mathtt{D}_2, \mathtt{D}_3\}$,
$\mathcal{A}_b := \{\mathtt{att}\}$,

and the set $\mathcal{T}_b$ of assertions consists of:

$$\begin{array}{lcl}
\mathtt{C}_0 & \dot{\preceq} & \exists\mathtt{att}.\mathtt{D}_1 \sqcap \exists\mathtt{att}.\mathtt{D}_2 \sqcap \exists\mathtt{att}.\mathtt{D}_3 \sqcap \exists^{\leq 2}\mathtt{att} \sqcap \forall\mathtt{att}^{-}.\bot \\
\mathtt{D}_1 & \dot{\preceq} & \neg\mathtt{C}_0 \sqcap \forall\mathtt{att}.\bot \sqcap \forall\mathtt{att}^{-}.\mathtt{C}_0 \\
\mathtt{D}_2 & \dot{\preceq} & \neg\mathtt{C}_0 \sqcap \forall\mathtt{att}.\bot \sqcap \forall\mathtt{att}^{-}.\mathtt{C}_0 \\
\mathtt{D}_3 & \dot{\preceq} & \neg\mathtt{C}_0 \sqcap \forall\mathtt{att}.\bot \sqcap \forall\mathtt{att}^{-}.\mathtt{C}_0
\end{array}$$

The assertions in $\mathcal{S}$ involving universal quantification and disjointness are not relevant for the example to work, but have been added to keep the number of compound classes and attributes as small as possible.

The expansion $\widehat{\mathcal{S}} := (\mathcal{C}, \mathcal{A}, \widehat{\mathcal{F}}, \widehat{\mathcal{P}}, \widehat{\mathcal{T}})$ of $\mathcal{S}$ is given by

$\widehat{\mathcal{F}}_n := \{\mathtt{C}\} \cup \{\mathtt{D}_{i_1 i_2 i_3} \mid i_1, i_2, i_3 \in \{0,1\}\}$, where
$\mathtt{C} := \{\mathtt{C}_0\}$, and $\mathtt{D}_{i_1 i_2 i_3} := \{\mathtt{D}_j \mid i_j = 1, j \in \{1,2,3\}\}$
$\widehat{\mathcal{P}}_n := \{\mathtt{att}[\emptyset, \emptyset]\} \cup \{\mathtt{att}[\mathtt{C}, \mathtt{D}_{i_1 i_2 i_3}] \mid i_1, i_2, i_3 \in \{0,1\}\}$,

and the set $\widehat{\mathcal{T}}$ of compound assertions consists of:

$$\begin{array}{lcl@{\qquad\qquad}lcl}
\mathtt{C} & \dot{\preceq} & \exists\mathtt{att}.\mathtt{D}_1 & \mathtt{C} & \dot{\preceq} & \exists\mathtt{att}.\mathtt{D}_3 \\
\mathtt{C} & \dot{\preceq} & \exists\mathtt{att}.\mathtt{D}_2 & \mathtt{C} & \dot{\preceq} & \exists^{\leq 2}\mathtt{att}.
\end{array}$$

The system $\Psi'_{\mathcal{S}}$ of inequalities derived from the expansion as explained above, i.e. including the additional inequalities to handle the qualified existential quantification is the following

$$\begin{array}{rcl}
2{\cdot}\mathrm{Var}(\mathtt{C}) & \geq & \displaystyle\sum_{i_1,i_2,i_3 \in \{0,1\}} \mathrm{Var}(\mathtt{att}[\mathtt{C}, \mathtt{D}_{i_1 i_2 i_3}]) \\
\mathrm{Var}(\mathtt{C}) & \leq & \displaystyle\sum_{i_2,i_3 \in \{0,1\}} \mathrm{Var}(\mathtt{att}[\mathtt{C}, \mathtt{D}_{1 i_2 i_3}]) \\
\mathrm{Var}(\mathtt{C}) & \leq & \displaystyle\sum_{i_1,i_3 \in \{0,1\}} \mathrm{Var}(\mathtt{att}[\mathtt{C}, \mathtt{D}_{i_1 1 i_3}]) \\
\mathrm{Var}(\mathtt{C}) & \leq & \displaystyle\sum_{i_1,i_2 \in \{0,1\}} \mathrm{Var}(\mathtt{att}[\mathtt{C}, \mathtt{D}_{i_1 i_2 1}])
\end{array}$$

Let *Sol* be the assignment that assigns

- the value 2 to $\mathrm{Var}(\mathtt{C})$,
- the value 1 to the unknowns corresponding to the compound classes in $\{\mathtt{D}_{111}, \mathtt{D}_{100}, \mathtt{D}_{010}, \mathtt{D}_{001}\}$, and the compound attributes in $\{\mathtt{att}[\mathtt{C}, \mathtt{D}_{111}], \mathtt{att}[\mathtt{C}, \mathtt{D}_{100}], \mathtt{att}[\mathtt{C}, \mathtt{D}_{010}], \mathtt{att}[\mathtt{C}, \mathtt{D}_{001}]\}$, and
- the value 0 to all other unknowns.

Then *Sol* is indeed a solution of $\Psi'_{\mathcal{S}}$ which is acceptable and satisfies the additional condition 6.3 required for the existence of a corresponding finite model. Nevertheless one can easily verify that there is no finite model of $\mathcal{S}$ in which the number of instances of each compound class and attribute is the one specified by the value that *Sol* assigns to the corresponding unknown. ∎

The intuitive reason why this simple approach does not lead to the desired results in the case at hand, is that it relies on the uniformity of all objects that are instances of the same compound class. When setting up the system of inequalities we are in fact transforming local constraints on the number of connections that a single object may have into global constraints on the total number of connections of a certain type. The necessary differentiation is introduced by constructing the expansion. Once this is done, all instances of the same compound concept can be regarded as equivalent. This is also reflected in the construction of the model in the proof of Lemma 6.2.11. The approach works for the case of $\mathcal{LUNI}$-schemata, where no class expression can distinguish between different instances of the same compound class. If we use existential quantification, however, due to the increased expressivity it is not sufficient anymore to split the schema into compound classes to obtain a uniform behaviour of the instances. This leads us to introduce the notion of biexpansion of a schema, where we make a more fine-grained separation based also on the existence of connections of certain types.

6.4.3 Biexpansion of a Schema

In order to define the biexpansion of a schema we need some additional terminology. Let $\mathcal{A}^- := \{A^- \mid A \in \mathcal{A}\}$ and $\mathcal{L} := \mathcal{A} \cup \mathcal{A}^-$. We use $\mathcal{G}_{\mathcal{C},\mathcal{A}}$ to denote $2^{\mathcal{C}} \times (2^{2^{\mathcal{C}}})^{\mathcal{L}}$. An element of $\mathcal{G}_{\mathcal{C},\mathcal{A}}$ is called a *bicompound class*, and it is constituted by a pair in which the first element is a compound class, and the second element is a function that associates to each attribute and inverse attribute a set of compound classes. An element of $\mathcal{A} \times \mathcal{G}_{\mathcal{C},\mathcal{A}} \times \mathcal{G}_{\mathcal{C},\mathcal{A}}$ is called a *bicompound attribute*. $\widetilde{G}$ and $\widetilde{Q}$ range over bicompound classes and bicompound attributes respectively. We introduce two functions that allow us to refer to the components of bicompound classes: The function $\mathrm{cc} : \mathcal{G}_{\mathcal{C},\mathcal{A}} \to 2^{\mathcal{C}}$ returns the first component of a bicompound class, and the function $\mathrm{ccs} : \mathcal{G}_{\mathcal{C},\mathcal{A}} \times \mathcal{L} \to 2^{2^{\mathcal{C}}}$ returns for a bicompound class $\widetilde{G}$ and a link L the set of compound classes assigned to L by the second component of $\widetilde{G}$. In analogy to compound attributes we use $A[\widetilde{G}_1, \widetilde{G}_2]$ to denote a bicompound attribute $(A, \widetilde{G}_1, \widetilde{G}_2)$.

Definition 6.4.8 A *doubly expanded schema* is a tuple $\widetilde{\mathcal{S}} = (\mathcal{C}, \mathcal{A}, \widetilde{\mathcal{G}}, \widetilde{\mathcal{Q}}, \widetilde{\mathcal{T}})$, where

- $\mathcal{C}$ is a set of atomic classes.
- $\mathcal{A}$ is a set of atomic attributes.
- $\widetilde{\mathcal{G}} \subseteq 2^{\mathcal{C}} \times (2^{2^{\mathcal{C}}})^{\mathcal{L}}$ is a set of bicompound classes.
- $\widetilde{\mathcal{Q}} \subseteq \mathcal{A} \times \widetilde{\mathcal{G}} \times \widetilde{\mathcal{G}}$ is a set of bicompound attributes.
- $\widetilde{\mathcal{T}}$ is a set of *bicompound assertions*. Each such assertion has one of the forms

$$\begin{array}{lll} \widetilde{G} & \dot{\preceq} & \exists^{\geq m} L \qquad \text{or} \\ \widetilde{G} & \dot{\preceq} & \exists^{\leq n} L, \end{array}$$

  where $\widetilde{G} \in \widetilde{\mathcal{G}}$, $L \in \mathcal{L}$, m is a positive, and n a nonnegative integer. ∎

Where not specified otherwise, we assume $\widetilde{\mathcal{S}} = (\mathcal{C}, \mathcal{A}, \widetilde{\mathcal{G}}, \widetilde{\mathcal{Q}}, \widetilde{\mathcal{T}})$.

The intuition behind the definition of bicompoud classes and attributes is the following: If $\mathrm{cc}(\widetilde{G}) = \widehat{F}$, then all instances of $\widetilde{G}$ are also instances of $\widehat{F}$. Let o be such an instance. Then for each compound class $\widehat{F}'$ in $\mathrm{ccs}(\widetilde{G}, L)$, there is an instance o' of $\widehat{F}'$ connected to o via link L, while for every compound class not in $\mathrm{ccs}(\widetilde{G}, L)$ there is no such instance. In analogy to compound attributes, a bicompound attribute $A[\widetilde{G}_1, \widetilde{G}_2]$ is interpreted as the restriction of attribute A to the pairs whose first component is an instance of $\widetilde{G}_1$ and whose second component is an instance of $\widetilde{G}_2$.

More formally, the semantics of a doubly expanded schema $\widetilde{\mathcal{S}}$ is given by extending interpretations over $\mathcal{C}$ and $\mathcal{A}$ to bicompound classes and attributes. For simplicity of notation we make use of class expressions built by applying the $\mathcal{LCNI}$-constructors to both class names and compound classes. The interpretation of such expressions is analogous to the interpretation of ordinary class expressions, considering the semantics

of compound classes as defined in the previous section. The semantics of bicompound classes and attributes is then defined as follows:

$$\begin{aligned}\widetilde{G}^{\mathcal{I}} &:= (\mathrm{cc}(\widetilde{G}))^{\mathcal{I}} \cap \bigcap_{L\in\mathcal{L}} \left(\bigcap_{\widehat{F}\in\mathrm{ccs}(\widetilde{G},L)} (\exists L.\widehat{F})^{\mathcal{I}} \cap \bigcap_{\widehat{F}\in 2^{\mathcal{C}}\setminus\mathrm{ccs}(\widetilde{G},L)} (\neg\exists L.\widehat{F})^{\mathcal{I}} \right) \\ (A[\widetilde{G}_1,\widetilde{G}_2])^{\mathcal{I}} &:= A^{\mathcal{I}} \cap (\widetilde{G}_1^{\mathcal{I}} \times \widetilde{G}_2^{\mathcal{I}}).\end{aligned}$$

This definition together with the fact that different compound classes have disjoint extensions implies that two different bicompound classes also have disjoint extensions. The same observation holds for two different bicompound attributes $A[\widetilde{G}_1,\widetilde{G}_2]$ and $A[\widetilde{G}'_1,\widetilde{G}'_2]$ that refer to the same attribute A.

The following easy lemma, shows how to obtain the extensions of classes and attributes, given the extensions of all bicompound classes and bicompound attributes.

Lemma 6.4.9 *Let $C \in \mathcal{C}$ be a class name and $A \in \mathcal{A}$ be an attribute name. Then the following holds:*

$$\begin{aligned}C^{\mathcal{I}} &= \bigcup_{\widetilde{G}\in\mathcal{G}_{\mathcal{C},\mathcal{A}} \mid C\in\mathrm{cc}(\widetilde{G})} \widetilde{G}^{\mathcal{I}} \\ A^{\mathcal{I}} &= \bigcup_{\widetilde{Q}\in\{A\}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}} \widetilde{Q}^{\mathcal{I}}.\end{aligned}$$

Proof. The claim follows directly from the semantics of bicompound classes and attributes. □

A (finite) interpretation $\mathcal{I} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *satisfies* a bicompound assertion $\widetilde{G} \doteq E$, if $\widetilde{G}^{\mathcal{I}} \subseteq E^{\mathcal{I}}$. $\mathcal{I}$ is a *(finite) model* of $\widetilde{\mathcal{S}}$ if the following conditions hold:

- For each bicompound class $\widetilde{G} \in \mathcal{G}_{\mathcal{C},\mathcal{A}} \setminus \widetilde{\mathcal{G}}$, it holds that $\widetilde{G}^{\mathcal{I}} = \emptyset$.
- For each bicompound attribute $\widetilde{Q} \in (\mathcal{A}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}) \setminus \widetilde{\mathcal{Q}}$, it holds that $\widetilde{Q}^{\mathcal{I}} = \emptyset$.
- $\mathcal{I}$ satisfies all bicompound assertions in $\widetilde{\mathcal{T}}$.

Definition 6.4.10 A doubly expanded schema $\widetilde{\mathcal{S}} = (\mathcal{C}, \mathcal{A}, \widetilde{\mathcal{G}}, \widetilde{\mathcal{Q}}, \widetilde{\mathcal{T}})$ is called the *biexpansion* of a schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ if the following conditions hold:

- $\widetilde{\mathcal{G}}$ is the set of all $\mathcal{S}$*-consistent* bicompound classes, where a bicompound class $\widetilde{G} \in \mathcal{G}_{\mathcal{C},\mathcal{A}}$ is said to be $\mathcal{S}$-consistent, if
  - $\mathrm{cc}(\widetilde{G})$ is $\mathcal{S}$-consistent.
  - for every $L \in \mathcal{L}$, for every $\widehat{F} \in \mathrm{ccs}(\widetilde{G}, L)$, $\widehat{F}$ is $\mathcal{S}$-consistent.
  - for every $L \in \mathcal{L}$, for every $D \in \mathrm{cc}(\widetilde{G})$, if $(D \doteq \exists L.D') \in \mathcal{T}$, then there is a $\widehat{F} \in \mathrm{ccs}(\widetilde{G}, L)$ such that $D' \in \widehat{F}$.
  - for every $L \in \mathcal{L}$, for every $D \in \mathrm{cc}(\widetilde{G})$, if $(D \doteq \forall L.D') \in \mathcal{T}$, then for all $\widehat{F} \in \mathrm{ccs}(\widetilde{G}, L)$ it holds that $D' \in \widehat{F}$.
- $\widetilde{\mathcal{Q}}$ is the set of all $\mathcal{S}$*-consistent* bicompound attributes, where a bicompound attribute $A[\widetilde{G}_1, \widetilde{G}_2] \in \mathcal{A}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}$ is said to be $\mathcal{S}$-consistent, if
  - $\widetilde{G}_1$ and $\widetilde{G}_2$ are $\mathcal{S}$-consistent.
  - for every $D \in \mathrm{cc}(\widetilde{G}_1)$, if $(D \doteq \forall A.D') \in \mathcal{T}$, then $D' \in \mathrm{cc}(\widetilde{G}_2)$.
  - for every $D \in \mathrm{cc}(\widetilde{G}_2)$, if $(D \doteq \forall A^{-}.D') \in \mathcal{T}$, then $D' \in \mathrm{cc}(\widetilde{G}_1)$.
  - $\mathrm{cc}(\widetilde{G}_2) \in \mathrm{ccs}(\widetilde{G}_1, A)$.
  - $\mathrm{cc}(\widetilde{G}_1) \in \mathrm{ccs}(\widetilde{G}_2, A^{-})$.

- $\widetilde{\mathcal{T}}$ is the smallest set of bicompound assertions such that for every $\widetilde{G} \in \widetilde{\mathcal{G}}$
  - if for some $D \in \mathrm{cc}(\widetilde{G})$ there is an assertion $(D \dot{\preceq} \exists^{\geq m} L) \in \mathcal{T}$, then $\widetilde{\mathcal{T}}$ contains the bicompound assertion $\widetilde{G} \dot{\preceq} \exists^{\geq m_{max}} L$, where
$$m_{max} := \max\{m \mid \exists D \in \mathrm{cc}(\widetilde{G}) \colon (D \dot{\preceq} \exists^{\geq m} L) \in \mathcal{T}\}.$$
  - if for some $D \in \mathrm{cc}(\widetilde{G})$ there is an assertion $(D \dot{\preceq} \exists^{\leq n} L) \in \mathcal{T}$, then $\widetilde{\mathcal{T}}$ contains the bicompound assertion $\widetilde{G} \dot{\preceq} \exists^{\leq n_{min}} L$, where
$$n_{min} := \min\{n \mid \exists D \in \mathrm{cc}(\widetilde{G}) \colon (D \dot{\preceq} \exists^{\leq n} L) \in \mathcal{T}\}.$$

■

It is easy to see that the size of the biexpansion is doubly exponential in the size of the original schema. The following lemma shows that it can also be effectively constructed in double exponential time.

Lemma 6.4.11 *The biexpansion $\widetilde{\mathcal{S}}$ of $\mathcal{S}$ can be constructed in time which is doubly exponential in $|\mathcal{S}|$.*

Proof. Similar to the proof of Lemma 6.2.6. □

The following lemmata relate the (finite) models of a schema to the (finite) models of its biexpansion.

Lemma 6.4.12 *Let $\mathcal{S}$ be a schema and $\mathcal{I}$ a (finite) model of $\mathcal{S}$. Then for each $\mathcal{S}$-inconsistent bicompound class or bicompound attribute X, $X^{\mathcal{I}} = \emptyset$.*

Proof. Let $\widetilde{G} \in \mathcal{G}_{\mathcal{C},\mathcal{A}}$ be $\mathcal{S}$-inconsistent. Then (a) $cc(\widetilde{G})$ is $\mathcal{S}$-inconsistent, or (b) there is a link $L \in \mathcal{L}$ and a compound class $\widehat{F} \in \mathrm{ccs}(\widetilde{G}, L)$ such that $\widehat{F}$ is $\mathcal{S}$-inconsistent, or (c) there is a link $L \in \mathcal{L}$ and an assertion $(D \dot{\preceq} \exists L.D') \in \mathcal{T}$ such that $D \in \mathrm{cc}(\widetilde{G})$ and for all $\widehat{F} \in \mathrm{ccs}(\widetilde{G}, L)$, $D \notin \widehat{F}$, or (d) there is a link $L \in \mathcal{L}$ and an assertion $(D \dot{\preceq} \forall L.D') \in \mathcal{T}$ such that $D \in \mathrm{cc}(\widetilde{G})$ and for some $\widehat{F}_0 \in \mathrm{ccs}(\widetilde{G}, L)$ $D' \notin \widehat{F}_0$. In case (a), by Lemma 6.4.5 $(\mathrm{cc}(\widetilde{G}))^{\mathcal{I}} = \emptyset$, and therefore $\widetilde{G}^{\mathcal{I}} = \emptyset$. In case (b), by Lemma 6.4.5 $\widehat{F}^{\mathcal{I}} = \emptyset$, which implies $(\exists L.\widehat{F})^{\mathcal{I}} = \emptyset$. From $\widetilde{G}^{\mathcal{I}} \subseteq (\exists L.\widehat{F})^{\mathcal{I}}$ it follows that $\widetilde{G}^{\mathcal{I}} = \emptyset$. In case (c), assume by contradiction that $o \in \widetilde{G}^{\mathcal{I}}$ for some $o \in \Delta^{\mathcal{I}}$. Since $D \in \mathrm{cc}(\widetilde{G})$, $o \in D^{\mathcal{I}}$, and since $\mathcal{I}$ satisfies $D \dot{\preceq} \exists L.D'$, there is $o' \in D'^{\mathcal{I}}$ such that $(o, o') \in L^{\mathcal{I}}$. Since $D^{\mathcal{I}} \cap \widehat{F}^{\mathcal{I}} = \emptyset$ for all $\widehat{F} \in \mathrm{ccs}(\widetilde{G}, L)$, there is some $\widehat{F}_0 \in 2^{\mathcal{C}} \setminus \mathrm{ccs}(\widetilde{G}, L)$ such that $o' \in \widehat{F}_0^{\mathcal{I}}$. But then $o \in (\exists L.\widehat{F}_0)^{\mathcal{I}}$ in contradiction to $\widetilde{G}^{\mathcal{I}} \subseteq (\neg\exists L.\widehat{F}_0)^{\mathcal{I}}$. In case (d), assume by contradiction that $o \in \widetilde{G}^{\mathcal{I}}$ for some $o \in \Delta^{\mathcal{I}}$. Since $\widehat{F}_0 \in \mathrm{ccs}(\widetilde{G}, L)$ there is $o' \in \widehat{F}_0^{\mathcal{I}}$ such that $(o, o') \in L^{\mathcal{I}}$. From $D' \notin \widehat{F}_0$ we have that $o' \notin D'^{\mathcal{I}}$, and from $D \in \mathrm{cc}(\widetilde{G})$ we have that $o \in D^{\mathcal{I}}$. This contradicts the fact that $\mathcal{I}$ satisfies $D \dot{\preceq} \forall R.D'$.

Let $\widetilde{Q} = A[\widetilde{G}_1, \widetilde{G}_2] \in \mathcal{A}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}$ be $\mathcal{S}$-inconsistent. Then (a) $\widetilde{G}_1$ or $\widetilde{G}_2$ is $\mathcal{S}$-inconsistent, or (b) there is an assertion $(D \dot{\preceq} \forall A.D') \in \mathcal{T}$ such that $D \in \widetilde{G}_1$ and $D' \notin \widetilde{G}_2$, or (c) there is an assertion $(D \dot{\preceq} \forall A^-.D') \in \mathcal{T}$ such that $D \in \widetilde{G}_2$ and $D' \notin \widetilde{G}_1$, or (d) $\mathrm{cc}(\widetilde{G}_2) \notin \mathrm{ccs}(\widetilde{G}_1, A)$, or (e) $\mathrm{cc}(\widetilde{G}_1) \notin \mathrm{ccs}(\widetilde{G}_2, A^-)$. In case (a) $\widetilde{G}_1^{\mathcal{I}} = \emptyset$ or $\widetilde{G}_2^{\mathcal{I}} = \emptyset$, and therefore $\widetilde{Q}^{\mathcal{I}} \subseteq \widetilde{G}_1^{\mathcal{I}}\times\widetilde{G}_2^{\mathcal{I}} = \emptyset$. In case (b), assume by contradiction that $(o, o') \in \widetilde{Q}^{\mathcal{I}} \subseteq A^{\mathcal{I}}$. Then $o \in \widetilde{G}_1^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and $o' \in \widetilde{G}_2^{\mathcal{I}}$. Since $\mathcal{I}$ satisfies $D \dot{\preceq} \forall A.D'$, it follows that $o' \in D'^{\mathcal{I}}$. But $\widetilde{G}_2^{\mathcal{I}} \cap L'^{\mathcal{I}} = \emptyset$ gives rise to a contradiction. Case (c) can be treated in a similar way. In case (d), assume by contradiction that $(o, o') \in \widetilde{Q}^{\mathcal{I}}$. Since $\mathrm{cc}(\widetilde{G}_2) \notin \mathrm{ccs}(\widetilde{G}_1, P)$, $\widetilde{G}_1^{\mathcal{I}} \subseteq (\neg\exists A.\mathrm{cc}(\widetilde{G}_2))^{\mathcal{I}}$, contradicting $o \in \widetilde{G}_1^{\mathcal{I}}$ and $o' \in \widetilde{G}_2^{\mathcal{I}} \subseteq (\mathrm{cc}(\widetilde{G}_2))^{\mathcal{I}}$. Case (e) can be treated in a similar way. □

Lemma 6.4.13 *Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be a schema, $\widetilde{\mathcal{S}}$ its biexpansion, and $\mathcal{I}$ an interpretation over $\mathcal{C}$ and $\mathcal{A}$. Then $\mathcal{I}$ is a (finite) model of $\mathcal{S}$ if and only if it is a (finite) model of $\widetilde{\mathcal{S}}$.*

Proof. "⇒" Let $\mathcal{I}$ be a (finite) model of $\mathcal{S}$. Since $\widetilde{\mathcal{G}}$ contains exactly the set of $\mathcal{S}$-consistent bicompound classes, by Lemma 6.4.12, for a bicompound class $\widetilde{G} \in \mathcal{G}_{\mathcal{C},\mathcal{A}} \setminus \widetilde{\mathcal{G}}$ we have that $\widetilde{G}^{\mathcal{I}} = \emptyset$. The same argument holds for bicompound attributes. Let $(\widetilde{G} \dot{\preceq} \exists^{\geq m} L) \in \widetilde{\mathcal{T}}$. Then there is a $D \in \mathrm{cc}(\widetilde{G})$ such that $(D \dot{\preceq} \exists^{\geq m} L) \in \mathcal{T}$. Since $\mathcal{I}$ satisfies this assertion and $\widetilde{G}^{\mathcal{I}} \subseteq (\mathrm{cc}(\widetilde{G}))^{\mathcal{I}} \subseteq L^{\mathcal{I}}$, $\mathcal{I}$ satisfies also $\widetilde{G} \dot{\preceq} \exists^{\geq m} L$. A similar argument holds if $(\widetilde{G} \dot{\preceq} \exists^{\leq n} L) \in \widetilde{\mathcal{T}}$.

"⇐" Let $\mathcal{I}$ be a (finite) model of $\widetilde{\mathcal{S}}$ and assume by contradiction that there is an assertion $(D \dot{\preceq} E) \in \mathcal{T}$ not satisfied by $\mathcal{I}$. This means there is some $o \in \Delta^{\mathcal{I}}$ such that $o \in D^{\mathcal{I}}$ and $o \notin E^{\mathcal{I}}$. Let $\widetilde{G}_o$ be the (unique) bicompound class such that $o \in \widetilde{G}_o^{\mathcal{I}}$. Then we have that $D \in \mathrm{cc}(\widetilde{G}_o)$. Suppose $E = D'$. Since $o \notin D'^{\mathcal{I}}$, we have that $D' \notin \mathrm{cc}(\widetilde{G}_o)$. It follows that $\mathrm{cc}(\widetilde{G}_o)$ and therefore also $\widetilde{G}_o$ are not $\mathcal{S}$-consistent. By Lemma 6.4.12 this contradicts $o \in \widehat{F}_o^{\mathcal{I}}$. The case where $E = D_1 \sqcup D_2$ can be handled in a similar way. Suppose $E = \forall A.D'$. Since $o \notin (\forall A.D')^{\mathcal{I}}$ there is some $o' \in \Delta^{\mathcal{I}}$ such that $(o, o') \in A^{\mathcal{I}}$ and $o' \notin D'^{\mathcal{I}}$. Let $\widetilde{G}_{o'}$ be the (unique) bicompound class such that $o' \in \widetilde{G}_{o'}^{\mathcal{I}}$. Then $D' \notin \mathrm{cc}(\widetilde{G}_{o'})$ and therefore $A[\widetilde{G}_o, \widetilde{G}_{o'}]$ is not $\mathcal{S}$-consistent. By Lemma 6.4.12 this contradicts $(o, o') \in A^{\mathcal{I}}$, $o \in \widetilde{G}_o^{\mathcal{I}}$, and $o' \in \widetilde{G}_{o'}^{\mathcal{I}}$. The case where $E = \forall A^-.D'$ is handled in a similar way. Suppose $E = \exists L.D'$. Since $D \in \mathrm{cc}(\widetilde{G}_o)$, there is a $\widehat{F}_0 \in \mathrm{ccs}(\widetilde{G}_a, L)$ such that $D' \in \widehat{F}_0$. Therefore $\widetilde{G}^{\mathcal{I}} \subseteq (\exists L.\widehat{F}_0)^{\mathcal{I}} \subseteq (\exists L.D')^{\mathcal{I}}$, which together with $o \in \widetilde{G}_o^{\mathcal{I}}$ contradicts $o \notin (\exists L.D')^{\mathcal{I}}$. Suppose $E = \exists^{\geq m} L$. Since $D \in \mathrm{cc}(\widetilde{G}_o)$, $\widetilde{\mathcal{T}}$ contains an assertion $\widetilde{G}_o \dot{\preceq} \exists^{\geq m_{max}} L$, where $m_{max} \geq m$. From $o \notin \exists^{\geq m} L^{\mathcal{I}}$ it follows that $o \notin \exists^{\geq m_{max}} L^{\mathcal{I}}$, which contradicts the fact that $\mathcal{I}$ satisfies all assertions in $\widehat{\mathcal{T}}$. The case where $E = \exists^{\leq n} L$ is handled in a similar way. □

6.4.4 System of Inequalities Corresponding to a Schema

We are now ready to define a system of linear inequalities whose solutions of a certain type are related to the finite models of the schema.

Definition 6.4.14 Let $\mathcal{S}$ be a schema and $\widetilde{\mathcal{S}} = (\mathcal{C}, \mathcal{A}, \widetilde{\mathcal{G}}, \widetilde{\mathcal{Q}}, \widetilde{\mathcal{T}})$ its biexpansion. Then the system $\Psi_{\mathcal{S}} = (\mathcal{V}, \mathcal{D})$ *corresponding to* $\mathcal{S}$ is defined as follows:

- $\mathcal{V} := \mathcal{V}_{\widetilde{\mathcal{G}}} \cup \mathcal{V}_{\widetilde{\mathcal{Q}}}$ is a set of unknowns, where
  - $\mathcal{V}_{\widetilde{\mathcal{G}}} := \{\mathrm{Var}(\widetilde{G}) \mid \widetilde{G} \in \widetilde{\mathcal{G}}\}$ and
  - $\mathcal{V}_{\widetilde{\mathcal{Q}}} := \{\mathrm{Var}(\widetilde{Q}) \mid \widetilde{Q} \in \widetilde{\mathcal{Q}}\}$.
- $\mathcal{D}$ is a set of homogeneous linear inequalities over the unknowns in $\mathcal{V}$ constituted by the following inequalities:
  - for each $\widetilde{G} \in \widetilde{\mathcal{G}}$, for each $A \in \mathcal{A}$, for each $\widehat{F} \in \mathrm{ccs}(\widetilde{G}, A)$ the inequality
$$\mathrm{Var}(\widetilde{G}) \leq \sum_{A[\widetilde{G}, \widetilde{G}_2] \in \widetilde{\mathcal{Q}} \mid \mathrm{cc}(\widetilde{G}_2) = \widehat{F}} \mathrm{Var}(A[\widetilde{G}, \widetilde{G}_2]). \tag{6.6}$$
  - for each $\widetilde{G} \in \widetilde{\mathcal{G}}$, for each $A \in \mathcal{A}$, for each $\widehat{F} \in \mathrm{ccs}(\widetilde{G}, A^-)$ the inequality
$$\mathrm{Var}(\widetilde{G}) \leq \sum_{A[\widetilde{G}_1, \widetilde{G}] \in \widetilde{\mathcal{Q}} \mid \mathrm{cc}(\widetilde{G}_1) = \widehat{F}} \mathrm{Var}(A[\widetilde{G}_1, \widetilde{G}]). \tag{6.7}$$
  - for each bicompound assertion $(\widetilde{G} \dot{\preceq} \exists^{\geq m} L) \in \widetilde{\mathcal{T}}$ the inequality
$$m \cdot \mathrm{Var}(\widetilde{G}) \leq S(\widetilde{G}, L), \tag{6.8}$$
    where
$$\begin{aligned} S(\widetilde{G}, A) &:= \sum_{A[\widetilde{G}, \widetilde{G}_2] \in \widetilde{\mathcal{Q}}} \mathrm{Var}(A[\widetilde{G}, \widetilde{G}_2]) \\ S(\widetilde{G}, A^-) &:= \sum_{A[\widetilde{G}_1, \widetilde{G}] \in \widetilde{\mathcal{Q}}} \mathrm{Var}(A[\widetilde{G}_1, \widetilde{G}]). \end{aligned}$$
  - for each bicompound assertion $(\widetilde{G} \dot{\preceq} \exists^{\leq n} L) \in \widetilde{\mathcal{T}}$ the inequality
$$n \cdot \mathrm{Var}(\widetilde{G}) \geq S(\widetilde{G}, L). \tag{6.9}$$

■

Where not specified otherwise we assume $\Psi_{\mathcal{S}} = (\mathcal{V}, \mathcal{D})$. The following lemma gives a double exponential upper bound for the construction of $\Psi_{\mathcal{S}}$.

Lemma 6.4.15 *The system of inequalities $\Psi_{\mathcal{S}}$ can be constructed in time which is at most double exponential in $|\mathcal{S}|$.*

Proof. By Lemma 6.4.11 the biexpansion $\widetilde{\mathcal{S}}$ of $\mathcal{S}$ can be constructed in double exponential time in $|\mathcal{S}|$, and therefore it has also at most double exponential size in $|\mathcal{S}|$. $\Psi_{\mathcal{S}}$ contains at most one inequality for each bicompound assertion in $\widetilde{\mathcal{S}}$, and each such inequality contains a number of unknowns that is smaller than the number of bicompound classes and bicompound attributes together. Therefore $|\Psi_{\mathcal{S}}|$ is at most quadratic in $|\widetilde{\mathcal{S}}|$ and it can also be effectively constructed in time that is quadratic in $|\widetilde{\mathcal{S}}|$. Summing up we obtain that $\Psi_{\mathcal{S}}$ can be constructed in at most double exponential size in $|\mathcal{S}|$. □

6.4.5 Characterization of Finite Class Consistency

In order to establish a correspondence between the existence of particular solutions of $\Psi_{\mathcal{S}}$ and the existence of models of a schema, consider a simple normalized $\mathcal{LCNI}$-schema $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$, where $\mathcal{A}$ contains only one attribute name A. We prove a preliminary result which gives a sufficient condition for an interpretation $\mathcal{I}$ over $\mathcal{C}$ and $\mathcal{A}$ to be a model of $\mathcal{S}$. Let $\widetilde{\mathcal{S}} = (\mathcal{C}, \mathcal{A}, \widetilde{\mathcal{G}}, \widetilde{\mathcal{Q}}, \widetilde{\mathcal{T}})$ be the biexpansion of $\mathcal{S}$ and assume $\widetilde{\mathcal{G}} := \{\widetilde{G}_{11}, \ldots, \widetilde{G}_{1K_1}, \ldots, \widetilde{G}_{H1}, \ldots, \widetilde{G}_{HK_H}\}$, where $\mathrm{cc}(\widetilde{G}_{hk}) = \mathrm{cc}(\widetilde{G}_{hk'}) =: \widehat{F}_h$, for $h \in \{1, \ldots, H\}$, and $k, k' \in \{1, \ldots, K_H\}$. The bicompound attribute $A[\widetilde{G}_{hk}, \widetilde{G}_{h'k'}]$ is abbreviated with $\widetilde{A}_{hk,h'k'}$. We also use $\widetilde{g}_{hk}$ as an abbreviation for $\mathrm{Var}(\widetilde{G}_{hk})$, and $\widetilde{a}_{hk,h'k'}$ as an abbreviation for $\mathrm{Var}(\widetilde{A}_{hk,h'k'})$. With I_G we denote the set $\{(h,k) \mid h \in \{1, \ldots, H\}, k \in \{1, \ldots, K_h\}\}$ of all pairs of indexes of bicompound classes in $\widetilde{\mathcal{G}}$, and with I_Q we denote the set of quadruples of indexes (h,k,h',k') of bicompound attributes in $\widetilde{\mathcal{Q}}$, i.e. $I_Q := \{(h,k,h',k') \mid \widetilde{A}_{hk,h'k'} \in \widetilde{\mathcal{Q}}\}$. Let $\Psi_{\mathcal{S}}$ be the system of linear inequalities obtained from $\widetilde{\mathcal{S}}$ as specified in Section 6.4.4.

Lemma 6.4.16 *Let $\bar{\mathcal{G}} := \{\bar{g}_{hk} \mid (h,k) \in I_G\}$ and $\bar{\mathcal{Q}} := \{\bar{a}_{hk,h'k'} \mid (h,k,h',k') \in I_Q\}$ be sets of nonnegative integer numbers such that $\Psi_{\mathcal{S}}$ is satisfied when we substitute $\bar{g}_{hk}$ for $\widetilde{g}_{hk}$ and $\bar{a}_{hk,h'k'}$ for $\widetilde{a}_{hk,h'k'}$. A finite model $\mathcal{I}$ of $\widetilde{\mathcal{S}}$ exists where*

1. *$\sharp \widetilde{G}^{\mathcal{I}}_{hk} = \bar{g}_{hk}$, for $(h,k) \in I_G$, and*
2. *$\sharp \widetilde{A}^{\mathcal{I}}_{hk,h'k'} = \bar{a}_{hk,h'k'}$, for $(h,k,h',k') \in I_Q$,*

if and only if

$$\bar{a}_{hk,h'k'} \leq \bar{g}_{hk} \cdot \bar{g}_{h'k'}, \quad \text{for} \quad (h,k,h',k') \in I_Q \tag{6.10}$$

Proof. "⇐" Suppose condition 6.10 holds. We exhibit a finite model $\mathcal{I} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\widetilde{\mathcal{S}}$ with $\bar{g}_{hk}$ instances of $\widetilde{G}_{hk}$, for $(h,k) \in I_G$, and $\bar{a}_{hk,h'k'}$ instances of $\widetilde{A}_{hk,h'k'}$, for $(h,k,h',k') \in I_Q$. For each $(h,k) \in I_G$ such that $\bar{g}_{hk} \neq 0$, we introduce $\bar{g}_{hk}$ symbols $b^1_{hk}, \ldots, b^{\bar{g}_{hk}}_{hk}$. Let $\Delta^{\mathcal{I}}$ be the set of all symbols b^j_{hk}. For $h \in \{1, \ldots, H\}$ we define $\widehat{F}^{\mathcal{I}}_h := \{b^j_{hk} \mid k \in \{1, \ldots, K_h\}, j \in \{1, \ldots, \bar{g}_{hk}\}\}$. Note that this is always possible, since the extensions of compound classes are always disjoint, and since $\widehat{F}_h = \mathrm{cc}(\widetilde{G}_{hk})$ by definition. Figure 6.2 specifies how to assign to each bicompound class $\widetilde{G}_{hk}$ a set $\mathrm{Ext}(\widetilde{G}_{hk}) \subseteq \widehat{F}^{\mathcal{I}}_h$ and to A a set $\mathrm{Ext}(A) \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ of pairs of elements of $\Delta^{\mathcal{I}}$. We show that by putting $\widetilde{G}^{\mathcal{I}}_{hk} := \mathrm{Ext}(\widetilde{G}_{hk})$ and $\widetilde{A}^{\mathcal{I}}_{hk,h'k'} := \mathrm{Ext}(A) \cap (\mathrm{Ext}(\widetilde{G}_{hk}) \times \mathrm{Ext}(\widetilde{G}_{hk}))$ we indeed obtain a finite interpretation which is also a model of $\widetilde{\mathcal{S}}$.

In the rest of the proof, when we refer to a step, we implicitly mean a step in the construction of Figure 6.2. In order to show that $\mathcal{I}$ is a model of $\widetilde{\mathcal{S}}$, we have to ensure the following:

(a) Each pair in $\mathrm{Ext}(A)$ gets assigned an element of $\Delta^{\mathcal{I}}$ to both of its components.

(b) For each bicompound class $\widetilde{G} \in \mathcal{G}_{\mathcal{C},\mathcal{A}} \setminus \widetilde{\mathcal{G}}$, it holds that $\widetilde{G}^{\mathcal{I}} = \emptyset$, and for each bicompound attribute $\widetilde{Q} \in (\mathcal{A} \times \mathcal{G}_{\mathcal{C},\mathcal{A}} \times \mathcal{G}_{\mathcal{C},\mathcal{A}}) \setminus \widetilde{\mathcal{Q}}$, it holds that $\widetilde{Q}^{\mathcal{I}} = \emptyset$.

(c) There is no couple of pairs in $\mathrm{Ext}(A)$ that get assigned the same pair of elements of $\Delta^{\mathcal{I}}$ to both components.

1. 1. For each bicompound class $\widetilde{G} \in \mathcal{G}_{\mathcal{C},\mathcal{A}} \setminus \widetilde{\mathcal{G}}$, let $\mathrm{Ext}(\widetilde{G}) \leftarrow \emptyset$.
   2. For each bicompound attribute $\widetilde{Q} \in \mathcal{A}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}\times\mathcal{G}_{\mathcal{C},\mathcal{A}} \setminus \widetilde{\mathcal{Q}}$, let $\mathrm{Ext}(\widetilde{Q}) \leftarrow \emptyset$.
   3. For each $(h,k) \in I_G$, if $\bar{g}_{hk} = 0$, then let $\mathrm{Ext}(\widetilde{G}_{hk}) \leftarrow \emptyset$, else let $\mathrm{Ext}(\widetilde{G}_{hk}) \leftarrow \{b^1_{hk}, \ldots, b^{\bar{g}_{hk}}_{hk}\}$.
   4. For each $(h,k,h',k') \in I_Q$, if $\bar{a}_{hk,h'k'} = 0$, then $\mathrm{Ext}(\widetilde{A}_{hk,h'k'}) \leftarrow \emptyset$, else introduce $\bar{a}_{hk,h'k'}$ pairs, assign to each such pair p a label $\mathrm{lab}(p) := (h,k,h',k')$, and let $\mathrm{Ext}(\widetilde{A}_{hk,h'k'}) \leftarrow \{p \mid \mathrm{lab}(p) = (h,k,h',k')\}$. Let $\mathrm{Ext}(A)$ be the set of all introduced pairs.
2. 1. Sort the elements of $\mathrm{Ext}(A)$ so that
      - elements with different labels are sorted with respect to the usual lexicographic ordering on their labels, and
      - elements with the same label are sorted arbitrarily.
   2. For each $(h,k) \in I_G$, if $\bar{g}_{hk} \neq 0$ then
      1. Mark all elements $p \in \mathrm{Ext}(A)$ with $\mathrm{lab}(p) = (h,k,h',k')$ for some $(h',k') \in I_G$. Let M be the number of marked elements and let p_m, for $m \in \{1,\ldots,M\}$, denote the m-th marked element.
      2. For $m \leftarrow 1$ to M (for each marked element) do
         assign b^i_{hk} to the first component of p_m, where
         $i = 1 + (m-1) \bmod \bar{g}_{hk}$.
      3. Un-mark all elements of $\mathrm{Ext}(A)$;
3. 1. Sort the elements of $\mathrm{Ext}(A)$ so that
      - elements with different labels are sorted with respect to the lexicographic ordering on their labels obtained by considering the components of the label in the following order: 3, 4, 1, 2, and
      - elements with the same label are sorted in such a way that elements with the same first component (according to step 2.2.2) are contiguous.
   2. For each $(h',k') \in I_G$, if $\bar{g}_{h'k'} \neq 0$ then
      1. Mark all elements $p \in \mathrm{Ext}(A)$ with $\mathrm{lab}(p) = (h,k,h',k')$ for some $(h,k) \in I_G$. Let M be the number of marked elements and let p_m, for $m \in \{1,\ldots,M\}$, denote the m-th marked element.
      2. For $m \leftarrow 1$ to M (for each marked element) do
         assign $b^i_{h'k'}$ to the second component of p_m, where
         $i = 1 + (m-1) \bmod \bar{g}_{h'k'}$.
      3. Un-mark all elements of $\mathrm{Ext}(A)$.

Figure 6.2: Construction of a finite model of $\widetilde{\mathcal{S}}$

(d) $\mathcal{I}$ satisfies all assertions in $\widetilde{\mathcal{T}}$.

With respect to case (a), since condition 6.10 holds, if an unknown $\bar{g}_{hk} = 0$, then also $\bar{a}_{hk,h'k'} = \bar{a}_{h'k',hk} = 0$ for all $(h',k') \in I_G$ and in step 1.4 no pairs labeled with (h,k,h',k') or (h',k',h,k) are introduced. This means that each pair in $\mathrm{Ext}(A)$ gets marked exactly once in step 2.2.1 and exactly once in step 3.2.1 and therefore gets assigned in steps 2.2.2 and 3.2.2 an instance of some bicompound class to the first and the second component.

With respect to case (b), we first show that for each bicompound class $\widetilde{G}_{hk} \in \widetilde{\mathcal{G}}$, the elements assigned to $\mathrm{Ext}(\widetilde{G}_{hk})$ in step 1.3 are indeed instances of $\widetilde{G}_{hk}$ in $\mathcal{I}$. If $\bar{g}_{hk} = 0$ we are done, since no elements are assigned to $\mathrm{Ext}(\widetilde{G}_{hk})$. If $\bar{g}_{hk} > 0$ we have to show that for $j \in \{1,\ldots,\bar{g}_{hk}\}$ the following holds: (b1) for every compound class $\widehat{F} \in \mathrm{ccs}(\widetilde{G}_{hk}, A)$, there is an instance $c \in \widehat{F}^{\mathcal{I}}$ such that $(b^j_{hk}, c) \in \mathrm{Ext}(A)$, and (b2) for every compound class $\widehat{F}' \in \mathrm{ccs}(\widetilde{G}_{hk}, A^-)$, there is an instance $c' \in \widehat{F}'^{\mathcal{I}}$ such that $(c', b^j_{hk}) \in \mathrm{Ext}(A)$. With respect to case (b1), let $\widehat{F}_{h_0} \in \mathrm{ccs}(\widetilde{G}_{hk}, A)$. Since $\bar{\mathcal{G}}$ and $\bar{\mathcal{Q}}$ satisfy the inequalities 6.6, in step 1.4 at least $\bar{g}_{hk}$ pairs labeled with (h,k,h_0,k') for some k' are introduced. In step 2.1 these pairs are sorted such that they are contiguous and in the iteration of step 2.2.1 relative to (h,k) they are all marked. Therefore, in step 2.2.2, since the elements $b^1_{hk},\ldots,b^{\bar{g}_{hk}}_{hk}$ are assigned to the first component of consecutive tuples, each b^j_{hk} gets assigned to the first component of a tuple labeled with (h,k,h_0,k'). In step 3.2.2 the second components of these tuples get assigned elements of $\mathrm{Ext}(\widetilde{G}_{h'_0k})$ and these are instances of $\widehat{F}_{h_0}$ by definition. Case (b2) can be shown in a similar way. This, together with the fact that all elements in $\Delta^{\mathcal{I}}$ get assigned to $\mathrm{Ext}(\widetilde{G}_{hk})$ for exactly one $(h,k) \in I_G$ and that bicompound classes are pairwise disjoint, implies that $\mathrm{Ext}(\widetilde{G}) = \widetilde{G}^{\mathcal{I}}$ for all bicompound classes in $\mathcal{G}_{\mathcal{C},\mathcal{A}}$. With respect to bicompound attributes, in steps 2.2.2 and 3.2.2 a pair labeled with (h,k,h',k') gets assigned to the first component only elements of $\mathrm{Ext}(\widetilde{G}_{hk}) = \widetilde{G}^{\mathcal{I}}_{hk}$ and to the second component only elements of $\mathrm{Ext}(\widetilde{G}_{h'k'}) = \widetilde{G}^{\mathcal{I}}_{h'k'}$. This implies that $\mathrm{Ext}(\widetilde{Q}) = \widetilde{Q}^{\mathcal{I}}$ for every bicompound attribute $\widetilde{Q}$. Step 1.1 ensures for each bicompound class $\widetilde{G} \in \mathcal{G}_{\mathcal{C},\mathcal{A}} \setminus \widetilde{\mathcal{G}}$, that $\mathrm{Ext}(\widetilde{G}) = \widetilde{G}^{\mathcal{I}} = \emptyset$. Similarly, step 1.2 ensures for each bicompound attribute $\widetilde{Q} \in (\mathcal{A}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}\times\mathcal{G}_{\mathcal{C},\mathcal{A}}) \setminus \widetilde{\mathcal{Q}}$, that $\mathrm{Ext}(\widetilde{Q}) = \widetilde{Q}^{\mathcal{I}} = \emptyset$.

With respect to case (c), notice that in step 2.2.2 an element b^j_{hk} can be assigned to the first component of a pair p only if $\mathrm{lab}(p) = (h,k,h',k')$ for some $(h',k') \in I_G$. A similar observation holds for the second component of p. This means that two pairs of $\mathrm{Ext}(A)$ can never get assigned the same pair of components if their labels are different. Therefore it is sufficient to show that for every $(h,k,h',k') \in I_Q$, two pairs with the same label (h,k,h',k') cannot get assigned the same pair of components. The number of pairs labeled with (h,k,h',k') is equal to $\bar{a}_{hk,h'k'}$. After the iteration of step 2.2 relative to the indexes (h,k), the largest group of pairs that get assigned the same element of $\mathrm{Ext}(\widetilde{G}_{hk})$ to their first component has at most

$$\rho_1 = \lceil \bar{a}_{hk,h'k'} / \bar{g}_{hk} \rceil \leq \bar{g}_{h'k'}$$

elements, where the inequality holds because of condition 6.10. Analogously, after the iteration of step 3.2 relative to the indexes (h',k'), the largest group of pairs that get assigned the same set of elements of $\mathrm{Ext}(\widetilde{G}_{hk})$ and $\mathrm{Ext}(\widetilde{G}_{h'k'})$ to the first and second components respectively, has at most

$$\rho_2 = \lceil \rho_1 / \bar{g}_{h'k'} \rceil \leq 1$$

elements.

With respect to case (d), consider an assertion $(\widetilde{G}_{hk} \dot{\preceq} \exists^{\geq m} A) \in \widetilde{\mathcal{T}}$. If $\bar{g}_{hk} = 0$, then in step 1.3 $\mathrm{Ext}(\widetilde{G}_{hk}) =: \widetilde{G}^{\mathcal{I}}_{hk}$ is set to $\emptyset$, and the assertion is trivially satisfied. Otherwise, since $\bar{g}_{hk}$ and all $\bar{a}_{hk,h'k'}$ satisfy inequalities 6.8 of $\Psi_{\mathcal{S}}$, it holds that

$$m \leq \left\lfloor \frac{\sum_{(h,k,h',k') \in I_Q} \bar{a}_{hk,h'k'}}{\bar{g}_{hk}} \right\rfloor \leq \left\lceil \frac{\sum_{(h,k,h',k') \in I_Q} \bar{a}_{hk,h'k'}}{\bar{g}_{hk}} \right\rceil .$$

On the other hand it is easy to see that for each element o^j_{hk} of $\mathrm{Ext}(\widetilde{G}_{hk})$, the number of pairs of $\mathrm{Ext}(A)$ that in step 2.2.2 get assigned o^j_{hk} to the first component is

$$\text{either} \quad \left\lfloor \frac{\sum_{(h,k,h',k')\in I_Q} \bar{a}_{hk,h'k'}}{\bar{g}_{hk}} \right\rfloor \quad \text{or} \quad \left\lceil \frac{\sum_{(h,k,h',k')\in I_Q} \bar{a}_{hk,h'k'}}{\bar{g}_{hk}} \right\rceil,$$

which shows that $\mathcal{I}$ satisfies the assertion. A similar argument shows that also assertions of the form $\widetilde{G}_{hk} \dot{\preceq} \exists^{\leq n} A$, or $\widetilde{G}_{hk} \dot{\preceq} \exists^{\geq m} A^-$, or $\widetilde{G}_{hk} \dot{\preceq} \exists^{\leq n} A^-$ in $\widetilde{\mathcal{T}}$ are satisfied in $\mathcal{I}$.

"$\Rightarrow$" Notice that the construction of the model must guarantee that $A^{\mathcal{I}}$ is a set and not a multiset, i.e. that no two pairs of $\mathrm{Ext}(A)$ get assigned the same first and second components. Suppose that for some $(h,k,h',k') \in I_Q$ condition 6.10 does not hold, i.e.

$$\bar{a}_{hk,h'k'} > \bar{g}_{hk} \cdot \bar{g}_{h'k'}. \tag{6.11}$$

Assume there is a finite model $\mathcal{I}$ of $\widetilde{\mathcal{S}}$ with $\bar{g}_{hk}$ instances of $\widetilde{G}_{hk}$, $\bar{g}_{h'k'}$ instances of $\widetilde{G}_{h'k'}$, and $\bar{a}_{hk,h'k'}$ different pairs in $P^{\mathcal{I}}$ whose first component is an instance of $\widetilde{G}_{hk}$ and whose second component is an instance of $\widetilde{G}_{h'k'}$. Since the number of different pairs that can be formed from $\bar{g}_{hk}$ different first components and $\bar{g}_{h'k'}$ different second components is $\bar{g}_{hk} \cdot \bar{g}_{h'k'}$, if condition 6.11 holds, at least two pairs in $P^{\mathcal{I}}$ must be equal, contradicting the assumption that such a model $\mathcal{I}$ exists. □

Similarly as for primitive $\mathcal{LUNI}$-schemata, the solutions of $\Psi_{\mathcal{S}}$ that correspond to finite models of $\mathcal{S}$ can now be characterized as those that are $\mathcal{W}_{\widetilde{\mathcal{S}}}$-acceptable (according to Definition 6.1.3) with respect to a particular binary relation $\mathcal{W}_{\widetilde{\mathcal{S}}}$. Let

$$\mathcal{W}_{\widetilde{\mathcal{S}}} := \{(\mathrm{Var}(\widetilde{Q}), \mathrm{Var}(\widetilde{G})) \mid \widetilde{Q} = A[\widetilde{G}, \widetilde{G}_2] \vee \widetilde{Q} = A[\widetilde{G}_1, \widetilde{G}]\}.$$

Acceptable solutions of $\Psi_{\mathcal{S}}$ are defined exactly as in Definition 6.2.12, considering the new definition of relation $\mathcal{W}_{\widetilde{\mathcal{S}}}$.

We are now ready to prove the main result concerning finite consistency of a class in a free $\mathcal{LCNI}$-schema.

Theorem 6.4.17 *Let $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ be an $\mathcal{LCNI}$-schema, $\widetilde{\mathcal{S}} := (\mathcal{C}, \mathcal{A}, \widetilde{\mathcal{G}}, \widetilde{\mathcal{Q}}, \widetilde{\mathcal{T}})$ its expansion, $C \in \mathcal{C}$ a class name, and $\Psi_{\mathcal{S}} := (\mathcal{V}, \mathcal{D})$ the system of inequalities derived from $\mathcal{S}$. Then C is finitely consistent in $\mathcal{S}$ if and only if $\Psi^C_{\mathcal{S}} := (\mathcal{V}, \mathcal{D}^C)$, where*

$$\mathcal{D}^C := \mathcal{D} \bigcup \left\{ \sum_{\widetilde{G}\in\widetilde{\mathcal{G}} \mid C \in \mathrm{cc}(\widetilde{G})} Var(\widetilde{G}) \geq 1 \right\}$$

admits an acceptable nonnegative integer solution.

Proof. "$\Leftarrow$" Let *Sol* be an acceptable nonnegative integer solution. For each bicompound class and bicompound attribute $\widetilde{X}$, let $Sol(\widetilde{X})$ denote the value assigned by *Sol* to $\mathrm{Var}(\widetilde{X})$.

We can assume that for every bicompound attribute $A[\widetilde{G}_1, \widetilde{G}_2] \in \widetilde{\mathcal{Q}}$ the following condition holds:

$$Sol(A[\widetilde{G}_1, \widetilde{G}_2]) \leq Sol(\widetilde{G}_1) \cdot Sol(\widetilde{G}_2). \tag{6.12}$$

In fact, since *Sol* is an acceptable solution, if $Sol(\widetilde{G}_1) = 0$ or $Sol(\widetilde{G}_2) = 0$, then $Sol(A[\widetilde{G}_1, \widetilde{G}_2]) = 0$ holds. Assume now that $Sol(\widetilde{G}_1) \neq 0$ and $Sol(\widetilde{G}_2) \neq 0$ and condition 6.12 does not hold for some bicompound attribute $A[\widetilde{G}_1, \widetilde{G}_2]$. Then it is sufficient to multiply $\mathcal{S}$ to a suitably large integer constant to obtain another integer solution $\mathcal{S}'$ of $\Psi^C_{\mathcal{S}}$ such that condition 6.12 is satisfied for $A[\widetilde{G}_1, \widetilde{G}_2]$. Since the only non-homogeneous inequality of $\Psi^C_{\mathcal{S}}$ is positive, part 1 of Lemma 6.1.2 guarantees that $\mathcal{S}'$ is indeed a solution of $\Psi^C_{\mathcal{S}}$.

For every attribute name $A \in \mathcal{A}$, let $\mathcal{S}_A = (\mathcal{C}, \mathcal{A}, \mathcal{T}_A)$ be the schema where $\mathcal{T}_A$ is obtained from $\mathcal{T}$ by removing all assertions involving an attribute different from A. Let $\widetilde{\mathcal{S}}$ be the biexpansion of $\mathcal{S}$ and $\widetilde{\mathcal{S}}_A$ the biexpansion of $\mathcal{S}_A$. Since $\mathcal{T}_A \subseteq \mathcal{T}$, each inequality of $\Psi_{\mathcal{S}_A}$ can be obtained from an inequality of $\Psi_{\mathcal{S}}$ by

removing the unknowns corresponding to bicompound classes and bicompound attributes present in $\widetilde{\mathcal{S}}_A$ but not present in $\widetilde{\mathcal{S}}$. Therefore *Sol* is also a solution of $\Psi_{\mathcal{S}_A}$ if it is extended by assigning the value 0 to all additional unknowns in $\Psi_{\mathcal{S}_A}$.

Lemma 6.4.16 ensures that for every attribute $A \in \mathcal{A}$, a finite model $\mathcal{I}_A$ of $\widetilde{\mathcal{S}}_A$ exists, with $Sol(\widetilde{Q})$ instances of $\widetilde{Q}$, for every $\mathcal{S}$-consistent bicompound attribute $\widetilde{Q}$ corresponding to A, and $Sol(\widetilde{G})$ instances of $\widetilde{G}$, for every $\mathcal{S}$-consistent bicompound class $\widetilde{G}$ of $\widetilde{\mathcal{S}}$. A finite model $\mathcal{I}$ of $\widetilde{\mathcal{S}}$ can easily be obtained by merging the models $\mathcal{I}_A$ for all attributes A and unifying the sets of instances of each bicompound class. This can in fact be done since all models of the biexpansions $\widetilde{\mathcal{S}}_A$ are constructed from the same solution of $\Psi_{\mathcal{S}}^C$ and therefore have the same number of instances for each bicompound class. By Lemma 6.4.13, $\mathcal{I}$ is also a model of $\mathcal{S}$. Moreover, since $\sharp\widetilde{G}^{\mathcal{I}} = Sol(\widetilde{G})$ for every bicompound class $\widetilde{G}$ and since $\sum_{\widetilde{G}|C\in\mathrm{cc}(\widetilde{G})} Sol(\widetilde{G}) \geq 1$, by applying Lemma 6.4.9 we have that

$$C^{\mathcal{I}} = \bigcup_{\widetilde{G}|C\in\mathrm{cc}(\widetilde{G})} \widetilde{G}^{\mathcal{I}} \neq \emptyset.$$

We can conclude that C is finitely consistent in $\mathcal{S}$.

"$\Rightarrow$" Suppose a finite model $\mathcal{I}$ of $\mathcal{S}$ exists, where $C^{\mathcal{I}} \neq \emptyset$. Let *Sol* be the solution that assigns to each unknown $\mathrm{Var}(\widetilde{X})$ the number of instances in $\mathcal{I}$ of the corresponding bicompound class or bicompound attribute $\widetilde{X}$, i.e. $Sol(\widetilde{X}) := \sharp\widetilde{X}^{\mathcal{I}}$. *Sol* is obviously integer and nonnegative. Moreover it is acceptable, since if a bicompound attribute $A[\widetilde{G}_1, \widetilde{G}_2]$ has a nonempty extension, then both $\widetilde{G}_1$ and $\widetilde{G}_2$ must have a nonempty extension, and therefore both $\mathcal{S}(\widetilde{G}_1) \neq 0$ and $\mathcal{S}(\widetilde{G}_2) \neq 0$.

We show that $\mathcal{S}$ satisfies all inequalities in $\Psi_{\mathcal{S}}^C$. With respect to inequalities 6.6, let $\widetilde{G} \in \widetilde{\mathcal{G}}$, $A \in \mathcal{A}$ and $\widehat{F} \in \mathrm{ccs}(\widetilde{G}, A)$. Since by definition $\widetilde{G}^{\mathcal{I}} \subseteq (\exists A.\widehat{F})^{\mathcal{I}}$, for each instance o of $\widetilde{G}$ in $\mathcal{I}$ there must be a pair $(o, o') \in A^{\mathcal{I}}$ such that o' is an instance of $\widehat{F}$ in $\mathcal{I}$. Therefore, the number of different pairs $(o_1, o_2) \in A^{\mathcal{I}}$ such that $o_1 \in \widetilde{G}^{\mathcal{I}}$ and $o_2 \in \widetilde{G}_2^{\mathcal{I}}$ for some bicompound class $\widetilde{G}_2$ with $\mathrm{cc}(\widetilde{G}_2) = \widehat{F}$ must be greater or equal to the number of instances of $\widetilde{G}$ in $\mathcal{I}$, which shows that inequalities 6.6 are satisfied. With respect to inequalities 6.7 we can proceed in a similar way. With respect to inequalities 6.8, let $(\widetilde{G} \dot{\preceq} \exists^{\geq m} A) \in \widetilde{\mathcal{T}}$. Since $\mathcal{I}$ is a model of $\mathcal{S}$, by Lemma 6.4.13, it is also a model of $\widetilde{\mathcal{S}}$ and therefore satisfies $\widetilde{G} \dot{\preceq} \exists^{\geq m} A$. It follows that for each instance p of $\widetilde{G}$ in $\mathcal{I}$ there must be $k \geq m$ different pairs $(o, o_1), \ldots, (o, o_k) \in P^{\mathcal{I}}$. Therefore, the number of pairs $(o', o'') \in A^{\mathcal{I}}$ such that $o' \in \widetilde{G}^{\mathcal{I}}$ must be at least m times the number of instances of $\widetilde{G}$ in $\mathcal{I}$, which shows that the inequality 6.8 corresponding to the above assertion is satisfied. With respect to the inequalities corresponding to a bicompound assertion involving an inverse attribute and with respect to inequalities 6.9 we can proceed in a similar way. Finally, since $C^{\mathcal{I}} \neq \emptyset$, there must be at least one bicompound class $\widetilde{G}$ such that $C \in \mathrm{cc}(\widetilde{G})$ and $\widetilde{G}^{\mathcal{I}} \neq \emptyset$. This shows that also the additional inequality in $\Psi_{\mathcal{S}}^C$ is satisfied. □

Since by Proposition 6.1.6 we can verify the existence of acceptable nonnegative integer solutions for a system of inequalities, we can make use of this result to effectively decide finite class consistency in free $\mathcal{LCNI}$- schemata.

6.4.6 Upper Bounds for Finite Model Reasoning

Summarizing the results of this section and using Proposition 6.1.6 we obtain the following theorems.

Theorem 6.4.18 *Finite class consistency $\mathcal{S} \not\models_f E \equiv \bot$ in free $\mathcal{LCNI}$-schemata can be decided in worst case deterministic double exponential time in $|\mathcal{S}| + |E|$.*

Proof. If E is not a class name, we construct the schema $\mathcal{S}' := (\mathcal{C}', \mathcal{A}, \mathcal{T}')$, where $\mathcal{C}' := \mathcal{C} \cup \{C\}$ with $C \notin \mathcal{C}$ and $\mathcal{T}' := \mathcal{T} \cup \{C \dot{\preceq} E, E \dot{\preceq} C\}$. It is easy to see that $\mathcal{S}$ and $\mathcal{S}'$ are equivalent, and that E is finitely consistent in $\mathcal{S}$ if and only if C is finitely consistent in $\mathcal{S}'$. Since $|\mathcal{S}'|$ is linear in $|\mathcal{S}| + |E|$, it is indeed sufficient to consider the case where E is a class name C.

By Lemma 6.4.1, we can construct a normalized schema $\mathcal{S}'$ equivalent to $\mathcal{S}$ such that $|\mathcal{S}'|$ is linear in $|\mathcal{S}|$. By Lemma 6.4.15, we can construct the corresponding system $\Psi_{\mathcal{S}'}^C$ of linear inequalities in time at most double exponential in $|\mathcal{S}|$ and therefore in $|\mathcal{S}'|$. By Theorem 6.4.17, C is finitely consistent in $\mathcal{S}'$, and by Lemma 2.3.7 in $\mathcal{S}$, if and only if $\Psi_{\mathcal{S}'}^C$ admits an acceptable nonnegative integer solution. By Proposition 6.1.6, the existence of such solution can be verified in polynomial time in $|\Psi_{\mathcal{S}'}^C|$, i.e. in worst case double exponential time in $|\mathcal{S}|$. □

Theorem 6.4.19 *Finite class subsumption $\mathcal{S} \models_f E_1 \preceq E_2$ in free $\mathcal{LCNI}$-schemata can be decided in worst case deterministic double exponential time in $|\mathcal{S}| + |E_1| + |E_2|$.*

Proof. Since $\mathcal{LCNI}$ contains general negation, for each $\mathcal{LCNI}$-class expression E, $\neg E$ is also an $\mathcal{LCNI}$-class expression. By Proposition 2.3.2, $\mathcal{S} \models_f E_1 \preceq E_2$ if and only if $\mathcal{S} \models_f E_1 \sqcap \neg E_2 \equiv \bot$, and by Theorem 6.4.18, this can be decided in worst case deterministic double exponential time in $|\mathcal{S}| + |E_1| + |E_2|$. □

6.5 Discussion

The reasoning technique proposed in Section 6.2 provides a tight upper bound for class consistency in primitive $\mathcal{LUNI}$-schemata. In fact, the method can easily be extended to handle a wider class of schemata, in which a negated class name and, more in general, a generic boolean combination of class names may appear on the left hand side of assertions. In particular, this makes it possible to deal also with schemata containg definitions of classes that are boolean combinations of class names, and reason on such schemata in deterministic exponential time.

The problematic constructor, which cannot be handled correctly by means of the expansion, is essentially qualifed existential quantification. We have seen that we can nevertheless reason on schemata in which such constructor may appear on the right hand side of a schema assertions, by paying the price of an additional exponential increase in size caused by the biexpansion. In fact, once we allow for qualified existential quantification, by the above observation and by Lemma 6.4.1, we can deal with arbitrary free assertions.

The method described in Section 6.4 can also be extended to schemata containing qualified number restrictions. To this end it is necessary to refine the biexpansion of the schema, since the differentiation introduced by bicompound concepts as specified in Definition 6.4.8 is too coarse. In fact, qualified number restrictions force us to make a separation in bicompound classes based not only on the existence but also on the number of links of a certain type. It turns out that it is in fact sufficient to consider only intervals of numbers of links, where the ranges of these intervals are given by the numbers that effectively appear in the schema. In this way it is still possible to keep the size of the resulting "extended" biexpansion doubly exponential in the size of the schema. The resulting "extended" bicompound assertions can then again be coded by means of a system of linear inequalities, and we can prove the counterpart of Theorem 6.4.17 for the system derived from a free $\mathcal{LCQI}$-schema.

The complexity gap between **EXPTIME**-hardness and the deterministic double exponential time algorithm for finite model reasoning in free $\mathcal{LCNI}$- and $\mathcal{LCQI}$-schemata is still open, however. A significant extension of the method would be provided by the possibility to handle, besides inverse, also other constructors on attributes, and especially transitive closure. This would make it possible to exploit reification of attributes and therefore to reason with respect to finite models in free $\mathcal{LT}^-$-schemata (not containing the repeat constructor).

Appendix A

Finite Automata on Infinite Objects

In this chapter we define the fundamental notions about finite automata on finite and infinite words and on infinite trees.

A.1 Sequential Automata

Sequential (finite state) automata, which accept regular sets of finite words over a specified alphabet, are omnipresent in Computer Science [96, 113].

Definition A.1.1 A *sequential automaton* is a tuple $A := (\Sigma, S, \delta, s_0, F)$, where

- Σ is a finite set of symbols called the *alphabet* of the automaton.
- S is a finite set of *states*.
- $\delta\colon S \times \Sigma \to 2^S$ is the *transition relation* that assigns to each state and letter the set of successor states.
- $s_0 \in S$ is the *initial* state.
- $F \subseteq S$ is the set of *final* states.

A is *deterministic* if the transition relation δ is deterministic, i.e $\sharp\delta(s,a) \leq 1$ for all $s \in S$ and all $a \in \Sigma$. δ is extended to Σ^* as follows (the empty string is denoted with ϵ):

$$\begin{aligned} \delta(s,\epsilon) &:= \{s\} \\ \delta(s,wa) &:= \{t \mid t \in \delta(s',a) \text{ for some } s' \in \delta(s,w)\}. \end{aligned}$$

A *accepts* a finite word $w \in \Sigma^*$ if $\delta(s_0,w) \cap F \neq \emptyset$. ∎

Büchi sequential automata are finite automata that have the same form as sequential automata, but use a different acceptance condition, since they accept infinite words rather than finite ones. They have been introduced by Büchi in [41], who was motivated by decision problems in mathematical logic. They have been shown to provide a normal form for certain monadic second order theories, and the decidability for these theories could be reduced to the decidability of the emptiness problem for Büchi automata. These results were then generalized by Rabin [132, 133] using automata on infinite trees.

We denote the set of infinite words over a finite alphabet Σ with Σ^ω. For an infinite word $w \in \Sigma^\omega$, $inf(w)$ is the set of letters that appear infinitely often in w. Note that since Σ is finite, $inf(w)$ is nonempty.

Definition A.1.2 A *Büchi sequential automaton* is a tuple $B := (\Sigma, S, \delta, s_0, F)$, where Σ, S δ, s_0, and F are as in Definition A.1.1.

A *run* of B over an infinite word $w := w_1 w_2 \cdots \in \Sigma^\omega$ is an infinite word $w_s := s'_0 s'_1 \cdots \in \mathcal{S}^\omega$, where $s'_0 = s_0$ and $s'_i \in \delta(s'_{i-1}, w_i)$, for $i \geq 1$. A run w_s of B over w is *accepting* if $inf(w_s) \cap F \neq \emptyset$. B *accepts* an infinite word $w \in \Sigma^\omega$ if it has an accepting run over w. ∎

Observe that the acceptance condition in a Büchi sequential automaton only cares about those states that occur infinitely often in a run, while nothing can be said about states that should appear finitely often.

The class of languages accepted by Büchi automata, called *ω-regular languages*, is closed under boolean operations (the closure under complement is extremely nontrivial [41]) and under concatenation of finite with infinite words. The class of *ω-regular languages* has also a natural definition in terms of basic operations on finite and infinite words, and corresponds in a precise sense to the class of regular languages on finite words.

However, if we require the transition relation of the automaton to be deterministic, Büchi acceptance condition is not sufficient to obtain the whole class of ω-regular languages (see for example [157]). In fact, deterministic Büchi automata are not closed under complement. This led to the introduction of variants of Büchi automata, in which the acceptance condition is changed and in which the restriction to deterministic transition relations represents no limitation in expressivity.

The following type of automata was introduced by Rabin [131]. The essential difference with respect to Büchi automata is that one can talk also about states that should occur only finitely often in an accepting run.

Definition A.1.3 A *Rabin sequential automaton* is a tuple $R := (\Sigma, S, \delta, s_0, \mathcal{F})$, where Σ, S δ, and s_0, are as in Definition A.1.1, and $\mathcal{F} \subseteq 2^S \times 2^S$ is a set of *accepting pairs*.

A run w_s of R over w is *accepting* if for some $(X, Y) \in \mathcal{F}$ we have $inf(w_s) \cap X = \emptyset$ and $inf(w_s) \cap Y \neq \emptyset$. R *accepts* an infinite word $w \in \Sigma^\omega$ if it has an accepting run over w. ∎

Büchi automata are just special cases of Rabin automata (with $\mathcal{F} := \{(\emptyset, F)\}$, using the notation of the respective definitions). Nevertheless Rabin sequential automata are not more powerful than Büchi sequential automata, since they also accept the class of ω-regular languages[1]. The usefulness of Rabin automata follows from the important determinization theorem by McNaughton [117], which implies that already deterministic Rabin sequential automata accept all ω-regular languages. While the determinization construction of McNaughton is rather complicated, an elegant essentially optimal construction was given in [135].

A.2 Tree Automata

For $x, y \in \{1, \ldots, n\}^*$ we say that x is a *prefix* of y if there is a $w \in \{1, \ldots, n\}^*$ such that $y = xw$. The *common prefix* of x and y is the longest prefix of both x and y. The following definition shows that for an infinite tree in which each node has exactly n successors the set of nodes can be identified with the elements of $\{1, \ldots, n\}^*$.

Definition A.2.1 An *(infinite) n-ary tree* T over an alphabet Σ is a labeling of the set $\{1, \ldots, n\}^*$ by elements of Σ, i.e. $T\colon \{1, \ldots, n\}^* \to \Sigma$. An element of $\{1, \ldots, n\}^*$ is a *node* of the tree. Let $x, y \in \{1, \ldots, n\}^*$.

- If $x = \epsilon$, then x is the *root* of the tree.
- If $y = xi$ for some $i \in \{1, \ldots, n\}$, then x is the *predecessor* of y and y is the i-th *successor* of x. We call the predecessor of x also 0-th *neighbor* of x, and the i-th successor of x also i-th *neighbor* of x.
- If x is a prefix of y, then x *precedes* y and y *succeeds* x, denoted with $x \preceq y$. x *properly precedes* y and y *properly succeeds* x, denoted with $x \prec y$, if $x \preceq y$ and $x \neq y$.
- A *path* starting at x is an infinite set $\{x_0, x_1, \ldots\}$ of nodes such that $x_0 = x$ and x_{i+i} is a successor of x_i, for all $i \geq 0$.

For a tree $T\colon \{1, \ldots, n\}^* \to \Sigma$ and a path p, $inf(T, p)$ is the set of labels that appear infinitely often on p. ∎

All types of automata on words introduced in Section A.1 can be generalized to trees: On a given n-ary tree T, the automaton starts its computation in an initial state at the root ϵ of the tree, and then simultaneously works down the paths of the tree level by level. The transition relation is therefore extended by specifying for the value of T in a certain node x and for each successor xi of x, the state assumed by the automaton in xi. We give here only the definitions of automata on infinite trees, which are the only ones that are used in this thesis.

[1] This is not the case for tree automata, where Rabin automata are strictly more powerful, as discussed in Section A.2.

Definition A.2.2 A *Büchi tree automaton* on n-ary trees is a tuple $B_t := (\Sigma, S, \delta, S_0, F)$, where

- Σ is the *alphabet* of the automaton.
- S is a finite set of *states*.
- $\delta\colon\ S \times \Sigma \to 2^{S^n}$ is the *transition relation* that assigns to each state and letter the set of possible n-tuples of successor states.
- $S_0 \subseteq S$ is the set of *initial* states.
- $F \subseteq S$ is the set of *final* states.

A *run* of B_t over an (infinite) n-ary tree $T\colon \{1,\ldots,n\}^* \to \Sigma$ is an n-ary tree $T_s\colon \{1,\ldots,n\}^* \to S$, where $T_s(\epsilon) \in S_0$ and for every $x \in \{1,\ldots,n\}^*$, we have that $(T_s(x1),\ldots,T_s(xn)) \in \delta(T_s(x), T(x))$. A run T_s of B_t over T is *accepting* if for every infinite path p starting at ϵ we have $inf(T_s, p) \cap F \neq \emptyset$.

B_t *accepts* an infinite tree T if it has an accepting run over T. ∎

Definition A.2.3 A *Rabin tree automaton* on n-ary trees is a tuple $R_t := (\Sigma, S, \delta, S_0, \mathcal{F})$, where Σ, S δ, and S_0, are as in Definition A.2.2, and $\mathcal{F} \subseteq 2^S \times 2^S$ is a set of accepting pairs.

A run T_s of R_t over T is *accepting* if for every infinite path p starting at ϵ there is some $(X, Y) \in \mathcal{F}$ such that $inf(T_s, p) \cap X = \emptyset$ and $inf(T_s, p) \cap Y \neq \emptyset$. R_t *accepts* an infinite tree T if it has an accepting run over T. ∎

These automata have been introduced and studied by Rabin [132, 133], and have been used to show one of the most profound decidability results in logic, namely the decidability of SnS, the monadic second order theory of n successors. Based on this result, a large number of problems in logic (including satisfiability in most propositional temporal and dynamic logics) could be shown to be decidable, by reducing them to SnS (see for example [81]).

Again, Büchi tree automata are a special case of Rabin tree automata. But different from the case of infinite words, Rabin tree automata are strictly more powerful than Büchi tree automata (see for example [157]).

The difference in expressivity is also reflected in a difference in the complexity of the emptiness problem for these automata, which has first been shown decidable in [133].

Theorem A.2.4 (Vardi and Wolper [165]) *The emptiness problem for Büchi tree automata is logspace complete for* ***PTIME****.* □

Theorem A.2.5 (Emerson and Jutla [72]) *The nonemptiness problem for Rabin tree automata is* ***NP****-complete. Moreover, there is an algorithm that, given a Rabin tree automaton R_t with n states and m accepting pairs, decides if the set of trees accepted by R_t is nonempty in time that is polynomial in n and exponential in m.* □

The latter result, together with Safra's construction for the determinization of sequential Büchi automata [135], has been proved fundamental for establishing tight upper bounds for satisfiability in a series of propositional and temporal logics whose decision procedures are based on automata on infinite trees [151, 164, 147, 152].

Appendix B

Propositional Dynamic Logics

Propositional Dynamic Logics (PDLs) have been introduced by Fischer and Ladner in [80] as a formal system for reasoning about computer programs and successively have been studied extensively and extended in several ways (see [107]).

B.1 Syntax and Semantics of PDLs

Syntactically, a PDL is constituted by expressions of two sorts: *programs* and *formulae*. Programs and formulae are built by starting from a set *Prog* of *program names* and a set *Prop* of *propositional letters* and applying suitable operators. We denote propositional letters with p, arbitrary formulae with f, program names with a, and arbitrary programs with r, all possibly with subscripts. We focus on the propositional dynamic logic CPDL [80], whose abstract syntax is as follows:

$$\begin{array}{rcl} f & \longrightarrow & \top \mid \bot \mid p \mid f_1 \wedge f_2 \mid \neg f \mid \langle r \rangle f \\ r & \longrightarrow & a \mid r_1 \cup r_2 \mid r_1; r_2 \mid r^* \mid r^- \mid f?. \end{array}$$

The propositional dynamic logic PDL is obtained from CPDL by eliminating the rule for converse programs.

We adopt the usual abbreviations for the other propositional connectives, and we identify a formula $\neg\neg f$ with f. We use $[r]f$ in place of $\neg\langle r\rangle\neg f$. With *atomic programs*, for which we use the letter b, we denote either program names or program names to which the converse operator is applied[1]. A *direct atomic program* is simply a program name and a *converse atomic program* is a program name to which the converse operator is applied. $Prog^-$ is the set of negative atomic programs. Formulae of the form $\langle r\rangle f$ are called *eventualities*, and programs of the form $f?$ are called *tests*.

In the following we assume without loss of generality that the converse operator is applied only to program names. This is justified by the following equivalences which are a consequence of the semantics given below:

$$\begin{array}{rclcrcl} (r_1; r_2)^- & \equiv & r_1^-; r_2^- & \qquad & (r^*)^- & \equiv & (r^-)^* \\ (r_1 \cup r_2)^- & \equiv & r_1^- \cup r_2^- & & (f?)^- & \equiv & f?. \end{array}$$

The semantics of PDLs (see for example [107]) is based on the notion of *(Kripke) structure*, which is defined as a triple $\mathcal{M} := (\mathcal{W}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \Pi^{\mathcal{M}})$, where $\mathcal{W}^{\mathcal{M}}$ is a non-empty (possibly infinite) set of *states* (or worlds), $\mathcal{R}^{\mathcal{M}} : Prog \to 2^{\mathcal{W}^{\mathcal{M}} \times \mathcal{W}^{\mathcal{M}}}$ is a *transition relation* that interprets program names, and $\Pi^{\mathcal{M}} : \mathcal{W}^{\mathcal{M}} \to 2^{Prop}$ assigns to each state the atomic propositions that are satisfied in that state. The transition relation $\mathcal{R}^{\mathcal{M}}$ can then be extended inductively to all programs as follows:

$$\begin{array}{rcl} \mathcal{R}^{\mathcal{M}}(r_1 \cup r_2) & := & \mathcal{R}^{\mathcal{M}}(r_1) \cup \mathcal{R}^{\mathcal{M}}(r_2) \\ \mathcal{R}^{\mathcal{M}}(r_1; r_2) & := & \mathcal{R}^{\mathcal{M}}(r_1) \circ \mathcal{R}^{\mathcal{M}}(r_2) \\ \mathcal{R}^{\mathcal{M}}(r^-) & := & \{(u, u') \in \mathcal{W}^{\mathcal{M}} \times \mathcal{W}^{\mathcal{M}} \mid (u, u') \in \mathcal{R}^{\mathcal{M}}(r)\} \\ \mathcal{R}^{\mathcal{M}}(r^*) & := & (\mathcal{R}^{\mathcal{M}}(r))^* \\ \mathcal{R}^{\mathcal{M}}(f?) & := & \{(u, u) \in \mathcal{W}^{\mathcal{M}} \times \mathcal{W}^{\mathcal{M}} \mid \mathcal{M}, u \models f\}, \end{array}$$

[1] For PDL atomic programs and program names coincide.

where *satisfaction* of a formula f in a state u, denoted with $\mathcal{M}, u \models f$, is defined inductively as follows:

- For $p \in Prop$, $\mathcal{M}, u \models p$ iff $p \in \Pi^{\mathcal{M}}(u)$.
- For all $u \in \mathcal{W}^{\mathcal{M}}$, $\mathcal{M}, u \models \top$.
- For no $u \in \mathcal{W}^{\mathcal{M}}$, $\mathcal{M}, u \models \bot$.
- $\mathcal{M}, u \models f_1 \wedge f_2$ iff $\mathcal{M}, u \models f_1$ and $\mathcal{M}, u \models f_2$.
- $\mathcal{M}, u \models \neg f$ iff not $\mathcal{M}, u \models f$.
- $\mathcal{M}, u \models \langle r \rangle f$ iff there exists a state u' such that $(u, u') \in \mathcal{R}^{\mathcal{M}}(r)$ and $\mathcal{M}, u' \models f$.

If the transition relation is required to be a function that assigns to each state and program a unique successor state, then we are dealing with the *deterministic* variants of PDL, called DPDL and CDPDL respectively [23, 165].

A structure $\mathcal{M}$ is called a *model* of a formula f if there exists a state $u \in \mathcal{W}^{\mathcal{M}}$ such that $\mathcal{M}, u \models f$. A formula f is *satisfiable* if there exists a model of f, unsatisfiable otherwise, and it is *valid* in structure $\mathcal{M}$, if for all $u \in \mathcal{W}^{\mathcal{M}}$, $\mathcal{M}, u \models f$. We call *axioms*, formulae that are assumed to be valid. Formally, a structure $\mathcal{M}$ is a model of an axiom f, if f is valid in $\mathcal{M}$[2]. An axiom is satisfiable, if it has a model. A structure $\mathcal{M}$ is a model of a finite set of axioms Γ, if $\mathcal{M}$ is a model of all axioms in Γ. A finite set of axioms is satisfiable if it has a model. We say that a finite set of axioms Γ *logically implies* a formula f, written $\Gamma \models f$, if f is valid in every model of Γ.

Observe that satisfiability of a formula f as well as satisfiability of a finite set of axioms Γ can be reformulated by means of logical implication as $\emptyset \not\models \neg f$ and $\Gamma \not\models \bot$, respectively.

The following theorem shows that axioms do not add expressivity to PDLs since they can be internalized in formulae. Therefore, when reasoning in PDLs it is sufficient to consider the problem of (un)satisfiability of a single PDL formula.

Theorem B.1.1 (Kozen and Tiuryn [107]) *Let Γ be a finite set of* CPDL*-axioms, and f a* CPDL*-formula. Then $\Gamma \models f$ if and only if the* CPDL*-formula*

$$\neg f \wedge [(b_1 \cup \ldots \cup b_h)^*]\Gamma'$$

is unsatisfiable, where $\{b_1, \ldots, b_h\}$ is the set $Prog \cup Prog^-$ of (direct and converse) atomic programs occurring in $\Gamma \cup \{f\}$ and Γ' is the conjunction of all axioms in Γ. □

This result exploits the power of program constructs (union, reflexive transitive closure) and the *connected model property*[3] of PDLs in order to represent axioms (valid formulae). An analogous result holds also for the most common extensions of CPDL. For the logics without converse operator, in the analogue of Theorem B.1.1 the formula to check for unsatisfiability is

$$\neg f \wedge [(a_1 \cup \ldots \cup a_h)^*]\Gamma'.$$

Theorem B.1.1 (and its analogues) is one of the main reasons to exploit the correspondence between $\mathcal{L}$-languages and PDLs.

B.2 The Correspondence between $\mathcal{L}$-Languages and PDLs

The correspondence between $\mathcal{L}$-languages and PDLs, first pointed out by Schild in [140], is based on the similarity between the interpretative structures of the two logics: at the extensional level, objects (members of $\Delta^{\mathcal{I}}$) in the $\mathcal{L}$-language correspond to states in PDL, whereas connections between two objects correspond to state transitions. At the intensional level, classes correspond to propositions, and attributes correspond to programs. The correspondence is established through a (one-to-one and onto) mapping δ from class expressions to PDL-formulae, and from attribute expressions to PDL-programs. We show the correspondence on the example of CPDL and $\mathcal{LCIL}$.

[2] Contrast this to the model of a formula, where the formula is only required to be satisfiable, i.e. to hold in one state.
[3] That is, if a formula has a model, it has a model which is connected.

Definition B.2.1 The mapping δ from $\mathcal{LCIL}$-class expressions to CPDL-formulae is defined inductively as follows:

$$\begin{array}{rcl@{\qquad}rcl} \delta(C) & := & C & \delta(A) & := & A \\ \delta(E_1 \sqcap E_2) & := & \delta(E_1) \wedge \delta(E_2) & \delta(L_1 \cup L_2) & := & \delta(L_1) \cup \delta(L_2) \\ \delta(E_1 \sqcup E_2) & := & \delta(E_1) \vee \delta(E_2) & \delta(L_1 \circ L_2) & := & \delta(L_1); \delta(L_2) \\ \delta(\neg E) & := & \neg\delta(E) & \delta(L^-) & := & \delta(L)^- \\ \delta(\forall L.E) & := & [\delta(L)]\delta(E) & \delta(L^*) & := & \delta(L)^* \\ \delta(\exists L.E) & := & \langle\delta(L)\rangle\delta(E) & \delta(id(E)) & := & \delta(E)? \end{array}$$

The mapping δ can then be extended to a mapping δ^+ from free $\mathcal{LCIL}$-schemata to CPDL-formulae. Namely, if $\mathcal{S} := (\mathcal{C}, \mathcal{A}, \mathcal{T})$ is an $\mathcal{LCIL}$-schema with $\mathcal{A} := \{A_1, \ldots, A_h\}$ and $\mathcal{T} := \{T_1, \ldots, T_k\}$, then

$$\begin{array}{rcl} \delta^+(\mathcal{S}) & := & [(A_1 \cup \cdots \cup A_h \cup A_1^- \cup \cdots \cup A_h^-)^*](\delta^+(T_1) \wedge \cdots \wedge \delta^+(T_k)) \\ \delta^+(E_1 \dot{\preceq} E_2) & := & (\delta(E_1) \rightarrow \delta(E_2)). \end{array}$$

■

The transformation δ together with the definitions of semantics for $\mathcal{L}$-languages and PDLs shows that these two formalisms are in fact just syntactic variant of each other. Mapping δ^+ carries this correspondence even further and allows to internalize assertions of $\mathcal{L}$-schemata in formulae of PDL. Therefore, by making use of Theorem B.1.1, both unrestricted class consistency and unrestricted class subsumption in $\mathcal{LCIL}$ can be (polynomially) reduced to (un)satisfiability of CPDL-formulae.

Theorem B.2.2 (Schild [140]) *Let $\mathcal{S}$ be a free $\mathcal{LCIL}$-schema and $\delta^+(\mathcal{S})$ the corresponding* CPDL*-formula. Then each model of $\mathcal{S}$ is isomorphic to a structure in which $\delta^+(\mathcal{S})$ is valid and vice versa. Moreover, in each connected structure, $\delta^+(\mathcal{S})$ is valid if and only if it is satisfiable.* □

Theorem B.2.3 (Schild [140]) *Let $\mathcal{S}$ be a free $\mathcal{LCIL}$-schema and E_1, E_2 two class expressions containing only class and attribute names of $\mathcal{S}$. Then*

1. *$\mathcal{S} \not\models_u E_1 \equiv \bot$ if and only if $\delta^+(\mathcal{S}) \wedge \delta(E_1)$ is satisfiable.*
2. *$\mathcal{S} \models_u E_1 \preceq E_2$ if and only if $\delta^+(\mathcal{S}) \wedge \delta(E_1) \wedge \neg\delta(E_2)$ is unsatisfiable.* □

Note that the size of the above CPDL-formulae is polynomial with respect to $|\mathcal{S}| + |E_1| + |E_2|$.

Since satisfiability for CPDL is an **EXPTIME**-complete problem (see [127]), we obtain the same complexity bounds for reasoning on free $\mathcal{LCIL}$-schemata with respect to unrestricted models.

Corollary B.2.4 *Unrestricted class consistency and class subsumption in free $\mathcal{LCIL}$-schemata are **EXPTIME**-complete problems.* □

The correspondence described above (through the mappings δ and δ^+) and the complexity results about reasoning can be extended to other $\mathcal{L}$-languages and PDLs in a straightforward way.

B.3 Extensions and Variants of PDLs

We introduce here some extensions and variants of PDLs.

B.3.1 Extensions of CPDL (RCPDL, QCPDL, FCPDL)

In this thesis we sometimes refer also to two extensions of CPDL. The first, called RCPDL, is obtained from CPDL by adding *repeat* formulae. Such formulae allow one to express infinite repetition of programs [151, 89] and are introduced by the syntax rule:

$$f \longrightarrow \Delta(r).$$

For a structure $\mathcal{M} := (\mathcal{W}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \Pi^{\mathcal{M}})$ and a state $u \in \mathcal{W}^{\mathcal{M}}$ we have that

- $\mathcal{M}, u \models \Delta(r)$ iff there is an infinite sequence $u_0, u_1, \ldots$ of states in $\mathcal{W}^{\mathcal{M}}$ such that $u_0 = u$, and $(u_i, u_{i+1}) \in \mathcal{R}^{\mathcal{M}}(r)$ for all $i \geq 0$.

The second extension we consider, called QCPDL, is obtained form CPDL by adding formulae which allow one to express *qualified number restrictions* on both direct and converse atomic programs in a state [56, 52]. Such formulae are introduced by the syntax rule:

$$f \longrightarrow \langle b \rangle^{\geq m} f' \mid \langle b \rangle^{\leq n} f',$$

where m is a positive and n a nonnegative integer.

For a structure $\mathcal{M} := (\mathcal{W}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \Pi^{\mathcal{M}})$ and a state $u \in \mathcal{W}^{\mathcal{M}}$ we have that

- $\mathcal{M}, u \models \langle b \rangle^{\geq m} f'$ iff there are at most m distinct states $u' \in \mathcal{W}^{\mathcal{M}}$ such that $(u, u') \in \mathcal{R}^{\mathcal{M}}(b)$ and $\mathcal{M}, u' \models f'$.
- $\mathcal{M}, u \models \langle b \rangle^{\leq n} f'$ iff there are at most m distinct states $u' \in \mathcal{W}^{\mathcal{M}}$ such that $(u, u') \in \mathcal{R}^{\mathcal{M}}(b)$ and $\mathcal{M}, u' \models f'$.

In this thesis we deal only with the restricted form of qualified number restrictions called *local functionality* [54] and described by

$$f \longrightarrow \langle a \rangle^{\leq 1} \mid \langle a^- \rangle^{\leq 1}.$$

We denote the extension of CPDL by this type of formulae with FCPDL.

FCPDL differs from CDPDL in two aspects: First, in FCPDL functionality for a program can be expressed locally, i.e. in one state, for a specified atomic program in contrast to global functionality for all program names of CDPDL. Second, FCPDL allows to express functionality of both direct and converse atomic programs, while no such restriction on converse atomic programs can be imposed in CDPDL. Since global functionality can easily be espressed by means of local functionality, FCPDL in fact properly extends CDPDL.

For the interested reader we also mention the tight relation that exists between qualified number restrictions and graded modalities in modal logic [158, 159, 76, 79].

The logic obtained form CPDL by adding both repeat formulae and qualified number restrictions (resp. local functionality) is called RQCPDL (resp. RFCPDL).

Formulae to express qualified number restricitions in PDLs correspond to qualified number restrictions in $\mathcal{L}$-languages, and the repeat constructor of $\mathcal{L}\Delta$ mimics its analogue in PDLs. Therefore the mapping δ from $\mathcal{L}$-languages to PDLs can easily be extended to these constructors and formulae:

$$\begin{aligned}
\delta(\Delta(L)) &:= \Delta(\delta(L)) \\
\delta(\exists^{\geq m} A.E) &:= \langle \delta(A) \rangle^{\geq m} \delta(E) \\
\delta(\exists^{\geq m} A^-.E) &:= \langle \delta(A^-) \rangle^{\geq m} \delta(E) \\
\delta(\exists^{\leq n} A.E) &:= \langle \delta(A) \rangle^{\leq n} \delta(E) \\
\delta(\exists^{\leq n} A^-.E) &:= \langle \delta(A^-) \rangle^{\leq n} \delta(E).
\end{aligned}$$

B.3.2 Variants of PDLs Described by Automata (APDLs)

The reasoning techniques we describe in the body of the thesis are presented in an automata-theoretic framework. For this reason it is useful to consider variants of PDLs in which the programs are described by finite state automata rather than by regular expressions [90, 128, 165]. We call such variants APDLs.

In an APDL, formulae are defined exactly as for the corresponding PDL, whereas programs are defined by the following syntax rule:

- If r is a sequential automaton over an alphabet Σ, where Σ is a finite set of atomic programs and tests, then r is a program.

We call a word $w \in \Sigma^*$ accepted by a program r (as defined in Section A.1) an *execution sequence of* r.

The semantics for formulae of an APDL is defined exactly as for the corresponding PDL, while the semantics for APDL programs is defined for atomic programs and test as for the corresponding PDL, and for programs expressed through automata by the following condition:

- $\mathcal{R}^{\mathcal{M}}(r) := \{(u, u') \mid$ there exists a word $w := w_1 \cdots w_n$ accepted by r and states $u_0, u_1, \ldots, u_n \in \mathcal{W}^{\mathcal{M}}$ such that $u = u_0$, $u' = u_n$ and for all $i \in \{1, \ldots, n\}$ we have $(u_{i-1}, u_i) \in \mathcal{R}^{\mathcal{M}}(w_i)\}$.

If the PDL we consider contains also repeat formulae, it is convenient to replace these by formulae expressed by means of a Büchi automaton, i.e. we replace the syntax rule for repeat with

$$f \longrightarrow \Delta(B),$$

where B is a Büchi sequential automaton over an alphabet Σ of atomic programs and tests. We call such formulae *Büchi-repeat* formulae.

The semantics for these formulae is based on Büchi acceptance condition for infinite words, as defined in Definition A.1.2. For a structure $\mathcal{M} := (\mathcal{W}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \Pi^{\mathcal{M}})$ and a state $u \in \mathcal{W}^{\mathcal{M}}$ we have that

- $\mathcal{M}, u \models \Delta(B)$ iff there exists an infinite word $w := w_1 w_2 \cdots$ accepted by B and an infinite sequence $u_0, u_1, \ldots$ of states in $\mathcal{W}^{\mathcal{M}}$ such that $u_0 = u$, and $(u_{i-1}, u_i) \in \mathcal{R}^{\mathcal{M}}(w_i)$ for all $i \geq 1$.

It has been shown in [161] that there is a linear translation from a PDL with repeat formulae to the corresponding PDL with Büchi repeat formulae (and a quadratic translation in the other direction). Since also the translation from regular expressions to automata is linear, all results that hold for an APDL can immediately be applied to the corresponding PDL. We assume also that the mappings δ and δ^+ of Definition B.2.1 are extended by a mapping ϕ that transforms each program appearing in an eventuality and expressed through a regular expression to a program expressed through an equivalent sequential automaton, and each repeat formula to an equivalent Büchi-repeat formula.

B.4 Additional Notions about PDLs

We introduce here some additional fundamental notions about PDLs that are at the basis of most results established for these logics.

B.4.1 Fischer-Ladner Closure

The notion of Fischer-Ladner closure introduced in [80] for PDL is useful when performing structural induction on formulae. Intuitively, the Fischer-Ladner closure of a formula f contains all "subformulae" of f that are relevant for verifying its satisfiability. It can be defined for all variants of PDLs, and we specify it here for APDLs.

Definition B.4.1 The *Fischer-Ladner closure* $CL(f_0)$ of a CAPDL-formula f_0 is the least set Φ of formulae such that:

- $f_0 \in \Phi$.
- If $\neg f \in \Phi$ then $f \in \Phi$.
- If $f \in \Phi$ then $\neg f \in \Phi$[4].
- If $f \wedge f' \in \Phi$ then $f, f' \in \Phi$.
- If $\langle r \rangle f \in \Phi$ then $f \in \Phi$.
- If $\langle r \rangle f \in \Phi$, where $r = (\Sigma, S, \delta, s_0, F)$, then $f' \in \Phi$ for all $f'? \in \Sigma$.
- If $\langle r \rangle f \in \Phi$, where $r = (\Sigma, S, \delta, s_0, F)$, then for all $s \in S$, $\langle r_s \rangle f \in \Phi$, where $r_s := (\Sigma, S, \delta, s, F)$ is the automaton obtained from r by making s to the initial state. ∎

It is easy to verify that $\sharp CL(f)$ is linear in $|f|$.

B.4.2 Tree Structures

Every structure can be ragarded as a graph by considering the states as nodes and the instances of the transition relations as arcs. A tree structure is a structure in which such graph has the form of an (infinite) tree of bounded degree. More formally it can be defined by considering trees as of Definition A.2.1.

Definition B.4.2 A structure $\mathcal{T} := (\mathcal{W}^{\mathcal{T}}, \mathcal{R}^{\mathcal{T}}, \Pi^{\mathcal{T}})$ is an (n-ary) *tree structure* if:

[4] We remind that we identify $\neg\neg f$ with f.

- $\mathcal{W}^{\mathcal{T}} \subseteq \{1, \ldots, n\}^*$.
- $xi \in \mathcal{W}^{\mathcal{T}}$ only if $x \in \mathcal{W}^{\mathcal{T}}$ and for some program name a either $(x, xi) \in \mathcal{R}^{\mathcal{T}}(a)$ or $(xi, x) \in \mathcal{R}^{\mathcal{T}}(a)$.
- $(x, y) \in \mathcal{R}^{\mathcal{T}}(a)$ for any program name a only if y is a neighbor of x, $(y, x) \notin \mathcal{R}^{\mathcal{T}}(a)$, and $(x, y) \notin \mathcal{R}^{\mathcal{T}}(a')$ and $(y, x) \notin \mathcal{R}^{\mathcal{T}}(a')$ for any other program name $a' \neq a$.

A structure is a *tree structure for a formula* f if it is an n-ary tree structure with n linear in $|f|$. ∎

When talking about tree structures we adopt the common terminology used for trees, as introduced in Definition A.2.1. We use also the term *b-arc*, where b is an atomic program, to denote a pair of nodes $(x, y) \in \mathcal{R}^{\mathcal{M}}(b)$. y is then said to be a *b-neighbor* of x.

The importance of tree structures in the context of PDLs is given by the fundamental *tree model property* shared by almost all variants of PDLs (and APDLs) considered in the literature. If a PDL has the tree model property, then every satisfiable formula of this PDL admits a model which is a tree structure for the formula. The following theorem states the tree model property for CAPDL.

Theorem B.4.3 (Streett [151]) *Let f be a satisfiable* CAPDL*-formula. Then f has an n-ary tree model $\mathcal{T}$, with $n \leq \sharp CL(f)$ and $\mathcal{T}, \epsilon \models f$.* □

The tree model property allows us to restrict our attention to tree structures when verifying satisfiability of a formula and makes it possible to use techniques based on automata on infinite trees [157]. In order to verify the satisfiability of a formula f one constructs a tree automaton that accepts exactly the tree structures for f. More precisely, the automaton accepts so called *Hintikka trees*, which are in direct correspondence with the tree structures for the formula. Therefore, in order to verify satisfiability of a formula it is sufficient to check whether the automaton associated to the formula accepts some (infinite) tree. Since such check can (usually) be performed in an efficient way [165, 72], the complexity of satisfiability in PDLs is buried in the construction of the automaton, whose size is exponential in the size of the formula.

By exploiting such techniques tight complexity bounds for satisfiability in a great variety of PDLs (and other logics of programs) have been obtained [151, 164, 152].

Bibliography

[1] Abiteboul, S., and Hull, R. IFO: A formal semantic database model. *ACM Transactions on Database Systems 12*, 4 (1987), 297–314.

[2] Abiteboul, S., and Kanellakis, P. Object identity as a query language primitive. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data* (1989), pp. 159–173.

[3] Abiteboul, S., Kanellakis, P., Ramaswamy, S., and Waller, E. Method schemas. Tech. Rep. CS-92-33, Brown University, 1992. An earlier version appeared in *Proc. of the 9th Symp. on Principles of Database Systems PODS-90*.

[4] Abiteboul, S., and Vianu, V. Generic computation and its complexity. In *Proc. of the 23th ACM SIGACT Sym. on Theory of Computing (STOC-91)* (1991), pp. 209–219.

[5] Abrial, J. R. Data semantics. In *Data Base Management*, J. W. Klimbie and K. L. Koffeman, Eds. North-Holland Publ. Co., Amsterdam, 1974, pp. 1–59.

[6] Albano, A., Ghelli, G., and Orsini, R. A relationship mechanism for strongly typed Object-Oriented database programming languages. In *Proc. of the 17th Int. Conf. on Very Large Data Bases (VLDB-91)* (Barcelona, 1991), pp. 565–575.

[7] Atzeni, P., and Parker Jr., D. S. Formal properties of net-based knowledge representation schemes. In *Proc. of the 2nd IEEE Int. Conf. on Data Engineering (ICDE-86)* (Los Angeles, 1986), pp. 700–706.

[8] Baader, F. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Tech. Rep. RR-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1990. An abridged version appeared in *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, pp. 446–451.

[9] Baader, F. Terminological cycles in KL-ONE-based knowledge representation languages. Tech. Rep. RR-90-01, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1990. An abridged version appeared in *Proc. of the 8th Nat. Conf. on Artificial Intelligence AAAI-90*, pp. 621–626.

[10] Baader, F. Terminological cycles in KL-ONE-based knowledge representation languages. In *Proc. of the 8th Nat. Conf. on Artificial Intelligence (AAAI-90)* (Boston, Ma., 1990), pp. 621–626.

[11] Baader, F. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)* (Sydney, Australia, 1991).

[12] Baader, F., Bürkert, H.-J., Heinsohn, J., Hollunder, B., Müller, J., Nebel, B., Nutt, W., and Profitlich, H.-J. Terminological knowledge representation: A proposal for a terminological logic. Tech. Rep. TM-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1991.

[13] Baader, F., and Hanschke, P. A schema for integrating concrete domains into concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)* (Sydney, 1991), pp. 452–457.

[14] Baader, F., and Hollunder, B. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge, PDK-91* (1991), no. 567 in Lecture Notes In Artificial Intelligence, Springer-Verlag, pp. 67–86.

[15] Baader, F., and Hollunder, B. Cardinality restrictions on concepts. Tech. Rep. RR-03-48, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 1993.

[16] Baader, F., and Hollunder, B. How to prefer more specific defaults in terminological default logic. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93)* (Chambery, France, 1993), Morgan Kaufmann, Los Altos, pp. 669–674.

[17] Baader, F., and Hollunder, B. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning 14* (1995), 149–180.

[18] Bancilhon, F., Delobel, C., and Kanellakis, P. *Building an Object-Oriented Database System – The story of O_2*. Morgan Kaufmann, Los Altos, 1992.

[19] Bancilhon, F., and Khoshafian, S. A calculus for complex objects. *Journal of Computer and System Sciences 38*, 2 (1989), 326–340.

[20] Batini, C., Ceri, S., and Navathe, S. B. *Conceptual Database Design, an Entity-Relationship Approach*. Benjamin and Cummings Publ. Co., Menlo Park, California, 1992.

[21] Beeri, C. A formal approach to object-oriented databases. *Data and Knowledge Engineering 5* (1990), 353–382.

[22] Beeri, C. New data models and languages – the challenge. In *Proc. of the 11th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-92)* (1992), pp. 1–15.

[23] Ben-Ari, M., Halpern, J. Y., and Pnueli, A. Deterministic propositional dynamic logic: Finite models, complexity, and completeness. *Journal of Computer and System Sciences 25* (1982), 402–417.

[24] Beneventano, D., Bergamaschi, S., Lodi, S., and Sartori, C. Using subsumption in semantic query optimization. In *IJCAI Workshop on Object-Based Representation Systems* (1993), A. Napoli, Ed.

[25] Bergamaschi, S., and Sartori, C. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems 17*, 3 (1992), 385–422.

[26] Berger, R. The undecidability of the dominoe problem. *Mem. Amer. Math. Soc. 66* (1966), 1–72.

[27] Bobrow, D. G., and Winograd, T. An overview of KRL, a Knowledge Representation Language. *Cognitive Science 1*, 1 (1977), 3–46. Republished in [33].

[28] Borgida, A. On the relationship between description logics and predicate logic queries. Tech. Rep. LCSR-TR-295-A, Rutgers University, New Brunswick, NJ, 1992.

[29] Borgida, A., Brachman, R. J., McGuinness, D. L., and Alperin Resnick, L. CLASSIC: A structural data model for objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data* (1989), pp. 59–67.

[30] Brachman, R. J. On the epistemological status of semantic networks. In *Associative Networks*, N. V. Findler, Ed. Academic Press, 1979, pp. 3–50.

[31] Brachman, R. J., Borgida, A., McGuinness, D. L., and Alperin Resnick, L. The CLASSIC knowledge representation system, or, KL-ONE: the next generation. Preprints of the Workshop on Formal Aspects of Semantic Networks, Two Harbors, Cal., 1989.

[32] Brachman, R. J., and Levesque, H. J. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI-84)* (1984), pp. 34–37.

[33] Brachman, R. J., and Levesque, H. J. *Readings in Knowledge Representation*. Morgan Kaufmann, Los Altos, 1985.

[34] Brachman, R. J., Pigman Gilbert, V., and Levesque, H. J. An essential hybrid reasoning system: Knowledge and symbol level accounts in KRYPTON. In *Proc. of the 9th Int. Joint Conf. on Artificial Intelligence (IJCAI-85)* (Los Angeles, 1985), pp. 532–539.

[35] Brachman, R. J., and Schmolze, J. G. An overview of the KL-ONE knowledge representation system. *Cognitive Science 9*, 2 (1985), 171–216.

[36] Buchheit, M., Donini, F. M., Nutt, W., and Schaerf, A. Terminological systems revisited: Terminology = schema + views. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)* (Seattle, USA, 1994), pp. 199–204.

[37] Buchheit, M., Donini, F. M., Nutt, W., and Schaerf, A. A refined architecture for terminological systems: Terminology = schema + views. Tech. Rep. RR-95-09, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 1995.

[38] Buchheit, M., Donini, F. M., and Schaerf, A. Decidable reasoning in terminological knowledge representation systems. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93)* (Chambery, France, 1993), Morgan Kaufmann, Los Altos, pp. 704–709.

[39] Buchheit, M., Donini, F. M., and Schaerf, A. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research 1* (1993), 109–138.

[40] Buchheit, M., Jeusfeld, M. A., Nutt, W., and Staudt, M. Subsumption between queries to Object-Oriented databases. *Information Systems 19*, 1 (1994), 33–54. Special issue on Extending Database Technology, EDBT'94.

[41] Büchi, J. R. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science* (1960), E. Nagel et al., Eds., Stanford University Press, pp. 1–11.

[42] Catarci, T., and Lenzerini, M. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems 2*, 4 (1993), 375–398.

[43] Cattell, R. G. G., Ed. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, Los Altos, 1994. Release 1.1.

[44] Chandra, A. K., and Harel, D. Computable queries for relational databases. *Journal of Computer and System Sciences 21* (1980), 156–178.

[45] Chen, P. P. The Entity-Relationship model: Toward a unified view of data. *ACM Transactions on Database Systems 1*, 1 (Mar. 1976), 9–36.

[46] Christophides, V., Abiteboul, S., Cluet, S., and Scholl, M. From structured documents to novel query facilities. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data* (Minneapolis (Minnesota, USA), 1994), R. T. Snodgrass and M. Winslett, Eds., pp. 313–324.

[47] Codd, E. F. A relational model of data for large shared data banks. *Communications of the ACM 13*, 6 (1970), 377–387.

[48] Collet, C., Huhns, M. N., and Shen, W. Resource integration using a large knowledge base in carnot. *IEEE Computer 24*, 12 (1991), 55–62.

[49] Cosmadakis, S. S., and Kanellakis, P. C. Functional and inclusion dependencies - A graph theoretical approach. In *Advances in Computing Research, Vol. 3*, P. C. Kanellakis and F. P. Preparata, Eds. JAI Press, 1986, pp. 163–184.

[50] Cosmadakis, S. S., Kanellakis, P. C., and Vardi, M. Polynomial-time implication problems for unary inclusion dependencies. *Journal of the ACM 37*, 1 (Jan. 1990), 15–46.

[51] Dalal, M. Tractable instances of some hard deduction problems. In *Proc. of the 10th European Conf. on Artificial Intelligence (ECAI-92)* (Vienna, 1992), John Wiley & Sons, pp. 354–363.

[52] De Giacomo, G. *Decidability of Class-Based Knowledge Representation Formalisms.* PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.

[53] De Giacomo, G. Eliminating converse from Converse PDL. *Journal of Logic, Language and Information* (1996). To appear.

[54] De Giacomo, G., and Lenzerini, M. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)* (1994), AAAI Press/The MIT Press, pp. 205–212.

[55] De Giacomo, G., and Lenzerini, M. Concept language with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. of the 11th European Conf. on Artificial Intelligence (ECAI-94)* (1994), pp. 411–415.

[56] De Giacomo, G., and Lenzerini, M. Converse, local determinism, and graded nondeterminism in propositional dynamic logics. Tech. Rep. 11-94, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", June 1994.

[57] De Giacomo, G., and Lenzerini, M. Description logics with inverse roles, functional restrictions, and n-ary relations. In *Proc. of the 4th European Workshop on Logics in Artificial Intelligence (JELIA-94)* (1994), vol. 838 of *Lecture Notes In Artificial Intelligence*, Springer-Verlag, pp. 332–346.

[58] De Giacomo, G., and Lenzerini, M. What's in an aggregate: Foundations for description logics with tuples and sets. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI-95)* (1995), pp. 801–807.

[59] Di Battista, G., and Lenzerini, M. Deductive entity-relationship modeling. *IEEE Transactions on Knowledge and Data Engineering 5*, 3 (1993), 439–450.

[60] Dionne, R., Mays, E., and Oles, F. J. A non-well-founded approach to terminological cycles. In *Proc. of the 10th Nat. Conf. on Artificial Intelligence (AAAI-92)* (1992), AAAI Press/The MIT Press, pp. 761–766.

[61] Donini, F. M., Hollunder, B., Lenzerini, M., Spaccamela, A. M., Nardi, D., and Nutt, W. The complexity of existential quantification in concept languages. Tech. rep., Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1990.

[62] Donini, F. M., Hollunder, B., Lenzerini, M., Spaccamela, A. M., Nardi, D., and Nutt, W. The complexity of existential quantification in concept languages. *Artificial Intelligence Journal 2–3* (1992), 309–327.

[63] Donini, F. M., Lenzerini, M., Nardi, D., and Nutt, W. The complexity of concept languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-91)* (1991), J. Allen, R. Fikes, and E. Sandewall, Eds., Morgan Kaufmann, Los Altos, pp. 151–162.

[64] Donini, F. M., Lenzerini, M., Nardi, D., and Nutt, W. Tractable concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)* (Sydney, 1991), pp. 458–463.

[65] Donini, F. M., Lenzerini, M., Nardi, D., and Nutt, W. The complexity of concept languages. Tech. Rep. RR-95-07, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 1995.

[66] Donini, F. M., Lenzerini, M., Nardi, D., Nutt, W., and Schaerf, A. Adding epistemic operators to concept languages. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-92)* (1992), Morgan Kaufmann, Los Altos, pp. 342–353.

[67] Donini, F. M., Lenzerini, M., Nardi, D., Nutt, W., and Schaerf, A. Queries, rules and definitions as epistemic sentences in concept languages. In *Proc. of the ECAI Workshop on Knowledge Representation and Reasoning* (1994), no. 810 in Lecture Notes In Artificial Intelligence, Springer-Verlag, pp. 113–132.

[68] Donini, F. M., Lenzerini, M., Nardi, D., Nutt, W., and Schaerf, A. Queries, rules and definitions. In *Foundations of Knowledge Representation and Reasoning* (1994), Springer-Verlag.

[69] Donini, F. M., Lenzerini, M., Nardi, D., and Schaerf, A. Deduction in concept languages: From subsumption to instance checking. *Journal of Logic and Computation 4*, 4 (1994), 423–452.

[70] Donini, F. M., Lenzerini, M., Nardi, D., and Schaerf, A. Reasoning in description logics. In *Foundation of Knowledge Representation*, G. Brewka, Ed. Cambrige University Press, 1996. To appear.

[71] Donini, F. M., Nardi, D., and Rosati, R. Non-first-order features of concept languages. In *Proc. of the 4th Conf. of the Italian Association for Artificial Intelligence (AI*IA-95)* (1995), Lecture Notes In Artificial Intelligence, Springer-Verlag.

[72] Emerson, E. A., and Jutla, C. S. The complexity of tree automata and logics of programs. In *Proc. of the 29nd Annual Sym. on the Foundations of Computer Science (FOCS-88)* (1988), pp. 328–337.

[73] Emerson, E. A., and Jutla, C. S. On simultaneously determinizing and complementing ω-automata. In *Proc. of the 4th Int. Conf. of Logic in Computer Science (LICS-89)* (1989), pp. 333–342.

[74] Emerson, E. A., and Sistla, A. P. Deciding full branching time logic. *Information and Control 61* (1984), 175–201.

[75] Fagin, R. Finite-model theory – a personal perspective. *Theoretical Computer Science 116* (1993), 3–31.

[76] Fattorosi-Barnaba, M., and Caro, F. D. Graded modalities I. *Studia Logica 44* (1985), 197–221.

[77] Ferg, S. Cardinality concepts in entity-relationship modeling. In *Proc. of the 10th Int. Conf. on the Entity-Relationship Approach (ER-91)* (1991), pp. 1–30.

[78] Fikes, R., and Kehler, T. The role of frame-based representation in reasoning. *Communications of the ACM 28*, 9 (1985), 904–920.

[79] Fine, K. In so many possible worlds. *Notre Dame Journal of Formal Logic 13*, 4 (1972), 516–520.

[80] Fischer, M. J., and Ladner, R. E. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences 18* (1979), 194–211.

[81] Gabbay, D. M. *Investigation in Modal and Tense Logics with Applications to Problems in Philosophy and Linguistics*. D. Reidel, 1976.

[82] Garey, M. R., and Johnson, D. S. *Computers and Intractability—A guide to NP-completeness*. W.H. Freeman and Company, San Francisco, 1979.

[83] Grant, J., and Minker, J. Numerical dependencies. In *Advances in Database Theory II*, H. Gallaire, J. Minker, and J.-M. Nicolas, Eds. Plenum Publ. Co., New York, 1984.

[84] Grant, J., and Minker, J. Inferences for numerical dependencies. *Theoretical Computer Science 41* (1985), 271–287.

[85] Gribomont, P., and Wolper, P. Temporal logic. In *From Modal Logic to Deductive Databases*, A. Thayse, Ed. John Wiley & Sons, 1989, ch. 4, pp. 165–233.

[86] Gurevich, Y. Logic and the challenge of computer science. In *Current Trends in Theoretical Computer Science*, E. Börger, Ed. Computer Science Press, Potomac, Maryland, 1988, ch. 1, pp. 1–57.

[87] Harel, D. Recurring dominoes: Making the highly undecidable highly understandable. In *Proc. of the Int. Conf. on Foundations of Computational Theory (FCT-83)* (1983), no. 158 in Lecture Notes in Computer Science, Springer-Verlag, pp. 177–194.

[88] Harel, D. Dynamic logic. In *Handbook of Philosophical Logic*, vol. 2. D. Reidel, Dordrecht, Holland, 1984, pp. 497–640.

[89] Harel, D., and Sherman, R. Looping vs. repeating in dynamic logic. *Information and Computation 55* (1982), 175–192.

[90] Harel, D., and Sherman, S. Propositional dynamic logic of flowcharts. In *Proc. of the Int. Conf. on Foundations of Computational Theory (FCT-83)* (1983), no. 158 in Lecture Notes in Computer Science, Springer-Verlag, pp. 195–206.

[91] Hayes, P. J. The logic of frames. In *Frame Conceptions and Text Understanding*, D. Metzing, Ed. Walter de Gruyter and Co., 1979, pp. 46–61. Republished in [33].

[92] Hidders, J., 1995. Personal communication.

[93] Hollunder, B. Hybrid inferences in KL-ONE-based knowledge representation systems. In *Proc. of the German Workshop on Artificial Intelligence* (1990), Springer-Verlag, pp. 38–47.

[94] Hollunder, B., and Baader, F. Qualifying number restrictions in concept languages. Tech. Rep. RR-91-03, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1991. An abridged version appeared in *Proc. of the 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning KR-91.*

[95] Hollunder, B., Nutt, W., and Schmidt-Schauss, M. Subsumption algorithms for concept description languages. In *Proc. of the 9th European Conf. on Artificial Intelligence (ECAI-90)* (London, 1990), Pitman, pp. 348–353.

[96] Hopcroft, J. E., and Ullman, J. D. *Formal Languages and their Relation to Automata.* Addison Wesley Publ. Co., Reading, Massachussetts, 1969.

[97] Hull, R. A survey of theoretical research on typed complex database objects. In *Databases*, J. Paredaens, Ed. Academic Press, 1988, pp. 193–256.

[98] Hull, R. B., and King, R. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys 19*, 3 (Sept. 1987), 201–260.

[99] Immermann, N. Relational queries computable in polynomial time. *Information and Computation 68* (1986), 86–104.

[100] Immermann, N. Languages which capture complexity classes. *SIAM Journal of Computing 16*, 4 (1987), 760–778.

[101] Kaczmarek, T. S., Bates, R., and Robins, G. Recent developments in NIKL. In *Proc. of the 5th Nat. Conf. on Artificial Intelligence (AAAI-86)* (1986), pp. 978–985.

[102] Karp, P. D. The design space of kowledge representation systems. Tech. Rep. SRI AI Technical Note 520, SRI International, Menlo Park, CA, 1992.

[103] Kerschberg, L., Klug, A., and Tsichritzis, D. A taxonomy of data models. In *Systems for Large Data Bases.* North-Holland Publ. Co., Amsterdam, 1976, pp. 43–64.

[104] Kim, W. *Introduction to Object-Oriented Databases.* The MIT Press, 1990.

[105] Kim, W. Object-oriented databases: Definitions and research directions. *IEEE Transactions on Knowledge and Data Engineering 2*, 3 (1990).

[106] Kozen, D. Results on the propositional μ-calculus. *Theoretical Computer Science 27* (1983), 333–354.

[107] Kozen, D., and Tiuryn, J. Logics of programs. In *Handbook of Theoretical Computer Science – Formal Models and Semantics* (1990), J. V. Leeuwen, Ed., Elsevier Science Publishers (North-Holland), Amsterdam, pp. 789–840.

[108] Lecluse, C., and Richard, P. Modeling complex structures in object-oriented databases. In *Proc. of the 8th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-89)* (1989), pp. 362–369.

[109] Lenzerini, M. Covering and disjointness constraints in type networks. In *Proc. of the 3rd IEEE Int. Conf. on Data Engineering (ICDE-87)* (Los Angeles, 1987), IEEE Computer, pp. 386–393.

[110] Lenzerini, M. Type data bases with incomplete information. *Information Sciences 53* (1991), 61–87.

[111] Lenzerini, M., and Nobili, P. On the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems 15*, 4 (1990), 453–461.

[112] Levesque, H. J. Knowledge representation and reasoning. *Annual Review of Computer Science 1* (1986), 255–287.

[113] Lewis, H. R., and Papadimitriou, C. H. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[114] MacGregor, R., and Bates, R. The Loom knowledge representation language. Tech. Rep. ISI/RS-87-188, University of Southern California, Information Science Institute, Marina del Rey, Cal., 1987.

[115] Mays, E., Dionne, R., and Weida, R. K-REP system overview. *SIGART Bulletin 2*, 3 (1991).

[116] McAllester, D., 1991. Unpublished Manuscript.

[117] McNaughton, R. Testing and generating infinite sequences by a finite automaton. *Information and Control 9* (1966), 521–530.

[118] Minsky, M. A framework for representing knowledge. In *Mind Design*, J. Haugeland, Ed. The MIT Press, 1981. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [33].

[119] Nebel, B. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence Journal 34*, 3 (1988), 371–383.

[120] Nebel, B. *Reasoning and Revision in Hybrid Representation Systems.* No. 422 in Lecture Notes In Artificial Intelligence. Springer-Verlag, 1990.

[121] Nebel, B. Terminological reasoning is inherently intractable. *Artificial Intelligence Journal 43* (1990), 235–249.

[122] Nebel, B. Terminological cycles: Semantics and computational properties. In *Principles of Semantic Networks*, J. F. Sowa, Ed. Morgan Kaufmann, Los Altos, 1991, pp. 331–361.

[123] Papadimitriou, C. H. On the complexity of integer programming. *Journal of the ACM 28*, 4 (1981), 765–768.

[124] Papadimitriou, C. H. *Computational Complexity.* Addison Wesley Publ. Co., Reading, Massachussetts, 1994.

[125] Patel-Schneider, P. F. Small can be beautiful in knowledge representation. In *Proc. of the IEEE Workshop on Knowledge-Based Systems* (1984). An extended version appeared as Fairchild Tech. Rep. 660 and FLAIR Tech. Rep. 37, October 1984.

[126] Peter D. Karp, K. L. M., and Gruber, T. The generic frame protocol. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI-95)* (Montreal, Canada, 1995), vol. A, pp. 768–774.

[127] Pratt, V. R. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences 20* (1980), 231–255.

[128] Pratt, V. R. Using graphs to understand PDL. In *Proc. of the 1st Workshop on Logics of Programs* (1981), no. 125 in Lecture Notes in Computer Science, Springer-Verlag, pp. 387–396.

[129] Quantz, J., and Kindermann, C. Implementation of the BACK system version 4. Tech. Rep. KIT-Report 78, FB Informatik, Technische Universität Berlin, Berlin, Germany, 1990.

[130] Quillian, M. R. Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science 12* (1967), 410–430. Republished in [33].

[131] Rabin, M. Automata on infinite objects and Church's problem. In *Proc. of Regional AMS Conf. Series in Mathematics* (1972), vol. 13, pp. 1–22.

[132] Rabin, M. O. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society 141* (1969), 1–35.

[133] Rabin, M. O. Weakly definable relations and special automata. In *Proc. of Symp. on Mathematical Logic and Foundations of Set Theory* (1970), Y. Bar-Hillel, Ed., North-Holland Publ. Co., Amsterdam, pp. 1–23.

[134] Robinson, R. Undecidability and nonperiodicity of tilings on the plane. *Inventiones Math. 12* (1971), 177–209.

[135] Safra, S. On the complexity of ω-automata. In *Proc. of the 29nd Annual Sym. on the Foundations of Computer Science (FOCS-88)* (1988), pp. 319–327.

[136] Sattler, U. A concept language for an engineering application with part-whole relations. In *Working Notes of the 1995 Description Logics Workshop* (Rome, Italy, 1995), A. Borgida, M. Lenzerini, D. Nardi, and B. Nebel, Eds., pp. 119–123. Technical Report, Università di Roma "La Sapienza", Dipartimento di Informatica e Sistemistica, RAP 07.95.

[137] Schaerf, A. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems 2* (1993), 265–278.

[138] Schaerf, A. *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues.* PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1994.

[139] Schaerf, A. Reasoning with individuals in concept languages. *Data and Knowledge Engineering 13*, 2 (1994), 141–176.

[140] Schild, K. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)* (Sydney, 1991), pp. 466–471.

[141] Schild, K. Terminological cycles and the propositional μ-calculus. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)* (Bonn, 1994), J. Doyle, E. Sandewall, and P. Torasso, Eds., Morgan Kaufmann, Los Altos, pp. 509–520.

[142] Schmidt-Schauss, M. Subsumption in KL-ONE is undecidable. In *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-89)* (1989), R. J. Brachman, H. J. Levesque, and R. Reiter, Eds., Morgan Kaufmann, Los Altos, pp. 421–431.

[143] Schmidt-Schauss, M., and Smolka, G. Attributive concept descriptions with complements. *Artificial Intelligence Journal 48*, 1 (1991), 1–26.

[144] Schmolze, J. G. Terminological knowledge representation systems supporting n-ary terms. In *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-89)* (1989), pp. 432–443.

[145] Schrijver, A. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.

[146] Shipman, D. W. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems 6*, 1 (1981), 140–173.

[147] Sistla, A. P., Vardi, M. Y., and Wolper, P. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science 49* (1987), 217–237.

[148] Smith, B. C. *Reflection and Semantics in a Procedural Language.* Ph.D. Thesis and Tech. Report MIT/LCS/TR-272, MIT, 1982. Prologue republished in [33].

[149] Staudt, M., Nissen, M., and Jeusfeld, M. Query by class, rule and concept. *Journal of Applied Intelligence 4*, 2 (1994), 133–157.

[150] Stirling, C. Modal and temporal logic. In *Handbook of Logic in Computer Science*, S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, Eds. Clarendon Press, Oxford, 1992, pp. 477–563.

[151] Streett, R. E. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Computation 54* (1982), 121–141.

[152] Streett, R. E., and Emerson, E. A. An automata theoretic decision procedure for the propositional μ-calculus. *Information and Computation 81* (1989), 249–264.

[153] Teorey, J. T. *Database Modeling and Design: The Entity-Relationship Approach*. Morgan Kaufmann, Los Altos, 1989.

[154] Teorey, T. J., Yang, D., and Frey, J. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys 18*, 2 (1986), 197–222.

[155] Thalheim, B. Fundamentals of cardinality constraints. In *Proc. of the 11th Int. Conf. on the Entity-Relationship Approach (ER-92)* (1992), G. Pernoul and A. M. Tjoa, Eds., Springer-Verlag, pp. 7–23.

[156] Thalheim, B. *Fundamentals of the Entity Relationship Model*. Springer-Verlag, 1993.

[157] Thomas, W. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, J. van Leuven, Ed., vol. B. Elsevier Science Publishers (North-Holland), Amsterdam, 1990, ch. 4, pp. 134–189.

[158] Van der Hoek, W. On the semantics of graded modalities. *Journal of Applied Non-Classical Logics 2*, 1 (1992), 81–123.

[159] Van Der Hoek, W., and De Rijke, M. Counting objects. *Journal of Logic and Computation 5*, 3 (1995), 325–345.

[160] Vardi, M. Y. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Sym. on Theory of Computing (STOC-82)* (1982), pp. 137–146.

[161] Vardi, M. Y. The taming of converse: Reasoning about two-way computations. In *Proc. of the 4th Workshop on Logics of Programs* (1985), R. Parikh, Ed., no. 193 in Lecture Notes in Computer Science, Springer-Verlag, pp. 413–424.

[162] Vardi, M. Y. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters 30*, 5 (1989), 261–264.

[163] Vardi, M. Y., and Stockmeyer, L. Improved upper and lower bounds for modal logics of programs: Preliminary report. In *Proc. of the 17th ACM SIGACT Sym. on Theory of Computing (STOC-85)* (1985), pp. 240–251.

[164] Vardi, M. Y., and Wolper, P. Automata-theoretic techniques for modal logics of programs. In *Proc. of the 16th ACM SIGACT Sym. on Theory of Computing (STOC-84)* (1984), pp. 446–455.

[165] Vardi, M. Y., and Wolper, P. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences 32* (1986), 183–221. A preliminary version appeared in *Proc. of the 16th ACM SIGACT Symp. on Theory of Computing STOC-84*.

[166] Vardi, M. Y., and Wolper, P. Reasoning about infinite computations. *Information and Computation 115*, 1 (1994), 1–37.

[167] Woods, W. A. What's in a link: Foundations for semantic networks. In *Representation and Understanding: Studies in Cognitive Science*, D. G. Bobrow and A. M. Collins, Eds. Academic Press, 1975, pp. 35–82. Republished in [33].

[168] Woods, W. A., and Schmolze, J. G. The KL-ONE family. In *Semantic Networks in Artificial Intelligence*, F. W. Lehmann, Ed. Pergamon Press, 1992, pp. 133–178. Published as a special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2–9.

[169] Ye, X., Parent, C., and Spaccapietra, S. Cardianlity consistency of derived objects in DOOD systems. In *Proc. of the 13th Int. Conf. on the Entity-Relationship Approach (ER-94)* (Manchester (UK), 1994), P. Loucopoulos, Ed., no. 881 in Lecture Notes in Computer Science, Springer-Verlag, pp. 278–295.

List of Publications

Calvanese, D. (1993). Finite satisfiability and implication in concept languages. In *Proc. of the Compulog Net annual area meeting on Knowledge Representation and Reasoning*. Universidade Nova de Lisboa.

Calvanese, D., & Lenzerini, M. (1994). Making object-oriented schemas more expressive. In *Proc. of the 13th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-94)*, pp. 243–254 Minneapolis. ACM Press and Addison Wesley.

Calvanese, D., & Lenzerini, M. (1994). On the interaction between ISA and cardinality constraints. In *Proc. of the 10th IEEE Int. Conf. on Data Engineering (ICDE-94)*, pp. 204–213 Houston. IEEE Computer Society Press.

Calvanese, D., Lenzerini, M., & Nardi, D. (1994). A unified framework for class based representation formalisms. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, pp. 109–120 Bonn. Morgan Kaufmann, Los Altos.

Calvanese, D., De Giacomo, G., & Lenzerini, M. (1995). Increasing the power of structured objects. In *KRDB-95: Reasoning about Structured Objects: Knowledge Representation meets Databases* Bielefeld (Germany).

Calvanese, D., De Giacomo, G., & Lenzerini, M. (1995). Structured objects: Modeling and reasoning. In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD-95)*, no. 1013 in Lecture Notes in Computer Science, pp. 229–246.

Calvanese, D. (1996). *Rappresentazione della conoscenza basata su classi: ragionamento in modelli finiti e in modelli arbitrari*. Ph.D. thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza". In Italian.

Calvanese, D. (1996). Reasoning with inclusion axioms in description logics: Algorithms and complexity. In Wahlster, W. (Ed.), *Proc. of the 12th European Conf. on Artificial Intelligence (ECAI-96)*, pp. 303–307. John Wiley & Sons. To appear.

Calvanese, D. (1996). Finite model reasoning in description logics. Submitted for publication.

Calvanese, D., & Lenzerini, M. (1996). Formal properties of ISA and cardinality constraints in data modeling. Submitted for publication.

Calvanese, D., & Lenzerini, M. (1996). Reasoning on object-oriented schemata. Manuscript.

Calvanese, D., Lenzerini, M., & Nardi, D. (1996). Foundations of class-based representation formalisms. Manuscript.

Dottorato di Ricerca in Informatica
Collana della tesi
Collection of Theses

V-93-1 Marco Cadoli. *Two Methods for Tractable Reasoning in Artificial Intelligence: Language Restriction and Theory Approximation.* June 1993.

V-93-2 Fabrizio d'Amore. *Algorithms and Data Structures for Partitioning and Management of Sets of Hyperrectangles.* June 1993.

V-93-3 Miriam Di Ianni. *On the complexity of flow control problems in Store-and-Forward networks.* June 1993.

V-93-4 Carla Limongelli. *The Integration of Symbolic and Numeric Computation by p-adic Construction Methods.* June 1993.

V-93-5 Annalisa Massini. *High efficiency self-routing interconnection networks.* June 1993.

V-93-6 Paola Vocca. *Space-time trade-offs in directed graphs reachability problem.* June 1993.

VI-94-1 Roberto Baldoni. *Mutual Exclusion in Distributed Systems.* June 1994.

VI-94-2 Andrea Clementi. *On the Complexity of Cellular Automata.* June 1994.

VI-94-3 Paolo Giulio Franciosa. *Adaptive Spatial Data Handling.* June 1994.

VI-94-4 Andrea Schaerf. *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues.* June 1994.

VI-94-5 Andrea Sterbini. *2-Thresholdness and its Implications: from the Synchronization with PVchunk to the Ibaraki-Peled Conjecture.* June 1994.

VII-95-1 Piera Barcaccia. *On the Complexity of Some Time Slot Assignment Problems in Switching Systems.* June 1995.

VII-95-2 Michele Boreale. *Process Algebraic Theories for Mobile Systems.* June 1995.

VII-95-3 Antonella Cresti. *Unconditionally Secure Key Distribution Protocols.* June 1995.

VII-95-4 Vincenzo Ferrucci. *Dimension-Independent Solid Modeling.* June 1995.

VII-95-5 Esteban Feuerstein. *On-line Paging of Structured Data and Multi-threaded Paging.* June 1995.

VII-95-6 Michele Flammini. *Compact Routing Models: Some Complexity Results and Extensions.* June 1995.

VII-95-7 Giuseppe Liotta. *Computing Proximity Drawings of Graphs.* June 1995.

VIII-96-1 Luca Cabibbo. *Querying and Updating Complex-Object Databases.* May 1996.

VIII-96-2 Diego Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms.* May 1996.

VIII-96-3 Marco Cesati. *Structural Aspects of Parameterized Complexity.* May 1996.

VIII-96-4 Flavio Corradini. *Space, Time and Nondeterminism in Process Algebras.* May 1996.

VIII-96-5 Stefano Leonardi. *On-line Resource Management with Application to Routing and Scheduling.* May 1996.

VIII-96-6 Rosario Pugliese. *Semantic Theories for Asynchronous Languages.* May 1996.