FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Normal Forms for Non-Relational Data

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Wolfgang Fischl
Matrikelnummer 0602106

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler
Mitwirkung: Univ.Ass. Dipl.Ing. Emanuel Sallinger
Univ.Ass. Dr.techn. Mantas Simkus, MSc
Univ.Prof. Dr. Diego Calvanese (Free University of Bolzano, Bozen)

Wien, 08.10.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Normal Forms for Non-Relational Data

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Wolfgang Fischl
Registration Number 0602106

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler
Assistance: Univ.Ass. Dipl.Ing. Emanuel Sallinger
Univ.Ass. Dr.techn. Mantas Simkus, MSc
Univ.Prof. Dr. Diego Calvanese (Free University of Bolzano, Bozen)

Vienna, 08.10.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Wolfgang Fischl
Setzgasse 32
7072 Mörbisch am See

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

In the first place I want to thank my advisor Reinhard Pichler for his guidance, support and inspiration during the time I was working on this thesis. I am also very grateful for his team at DBAI. In particular, I want to thank Emanuel Sallinger and Mantas Simkus for their great advice and ideas. They have spent an unbelievable amount of time discussing and finalizing this thesis with me. Furthermore, I am very glad that Diego Calvanese joined our team. His knowledge and insights on Description Logics was helpful in many areas.

During my time as a student at the University of Vienna and the Technical University of Vienna I met exceptionally supportive people that have driven my interest and knowledge in various scientific fields. As a representative for all who I have met, I want to thank Arndt von Haeseler and all members of his group at the Center for Integrative Bioinformatics Vienna for their support and friendship.

Last, but most importantly, my gratitude goes to my friends and my family. To my parents Helga and Martin, who have always encouraged me to pursue my personal goals and made my academic studies possible. To Kerstin, for her patience during the days in which it has not been easy with me. Especially, in situations where academia was more important to me than anything else, I have always encountered tolerance for my work.

Dankeschön.

This research is supported by the Austrian Science Fund (FWF):P25207-N23.

Abstract

The amount of data stored in today's information systems is increasing rapidly. Most widely used for this task are relational database management systems. However, alternative data formats, like XML documents or graph databases, continue to become more and more popular. In all these data formats database design is an important task to avoid redundancies arising from badly designed schemata. Therefore, Normal Forms were developed. Most prominently, Boyce-Codd Normal Form (BCNF) is used for relational models. Arenas and Libkin introduced 2004 XML Normal Form for XML documents. So far, a normal form for graph databases has not been considered yet. Our goal is to define a normal form that captures the intuition of BCNF for graph databases.

We will recall Boyce-Codd Normal Form and XML Normal Form and will then use ideas from these to define a normal form for graph databases. Description Logics (DLs) are ideally suited as a formal model for graph databases. Since BCNF is formulated over functional dependencies (FDs), we need to express FDs over DL knowledge bases (KBs). A first candidate are path-based identification constraints introduced by Calvanese in 2008. However, we show that path-based identification constraints are not powerful enough to model functional dependencies. Therefore, we propose tree-based identification constraints as an extension of path-based identification constraints. Based on tree-based identification constraints we look at redundancy in DLs.

The main result of this thesis is a definition of Description Logic Normal Form, which is a faithful translation of BCNF to Description Logics. Additionally, we introduce a direct mapping from relational schemas to DL KBs and show that if a relational schema is in BCNF, then the DL KB, directly mapped from this schema, is in DLNF and vice versa.

Kurzfassung

Die Menge an von Informationssystemen gespeicherten Daten wächst stetig. Für diesen Zweck werden vorwiegend Datenbanksysteme eingesetzt. Jedoch gewinnen alternative Datenformate, wie XML Dokumente und Graph-basierte Datenbanken, immer mehr an Einfluss. In all diesen Datenformaten ist es wichtig, mit Hilfe des Datenbankdesigns Redundanzen zu vermeiden. Solche können bei einem schlecht konzipierten Datenmodell auftreten. Deshalb wurden Normalformen entwickelt, um Datenmodelle zu schaffen, welche keine vermeidbaren Redundanzen mehr enthalten. Für das relationale Datenmodell wird Boyce-Codd Normalform (BCNF) verwendet. Arenas und Libkin entwickelten 2004 XML Normalform für XML Dokumente. Normalformen für Graph-basierte Datenbanken wurden bisher nicht untersucht. Unser Ziel ist es, diese Lücke zu schließen. Wir wollen eine Normalform definieren, welche die Eigenschaften von BCNF auf Graph-basierte Datenbanken überträgt.

Zuerst werden wir Boyce-Codd und XML Normalform betrachten. Ideen aus diesen Normalformen verwenden wir dazu, um eine Normalform für Graph-basierte Datenbanken zu entwickeln. Beschreibungslogiken (DLs) sind als formales Modell für Graph-basierte Datenbanken überaus passend. Da BCNF mittels funktionaler Abhängigkeiten definiert ist, muss es uns möglich sein solche auch über DL Wissensbasen (DL KBs) auszudrücken. Ein naheliegender Kandidat sind die von Calvanese et al. 2008 eingeführten path-based identification constraints verwenden. Allerdings zeigen wir, dass path-based identification constraints nicht ausdrucksstark genug sind, um funktionale Abhängigkeiten in DLs zu modellieren. Deshalb erweitern wir path-based identification constraints zu tree-based identification constraints. Mittels diesen untersuchen wir Redundanzen in DL KBs.

Der Hauptbeitrag dieser Arbeit ist eine Definition der Beschreibungslogik Normalform (DLNF), welche eine sinnesgetreue Erweiterung der BCNF zu DLs ist. Zusätzlich stellen wir eine direkte Abbildung von relationalen Schemata auf DL KBs vor. Wir zeigen, dass jedes relationale Schema genau dann in BCNF ist, wenn auch die direkte Abbildung dieses Schemas auf eine DL KB in DLNF ist und umgekehrt.

Contents

CHAPTER 1

Introduction

Database Management Systems (DBMS) are among the most widely used information systems. Their importance is unquestioned. The amount of data in DBMS is increasing rapidly. This success is largely due to the simplicity and elegance of the *relational model*. In the relational model [31], data is stored in relations or simply, tables. In addition to the relational data, other data models have been established over the past decade. With the development of the World Wide Web, the *Extensible Markup Language* (XML) has become a popular data model. In XML, data is stored in a tree-like structure [66, 67]. In recent years yet another data model, *graph databases* [50, 51], has gained popularity. Graph databases capture the inherent graph structure of data used in many different applications [2] such as the Semantic Web [45] as well as social and biological networks [11].

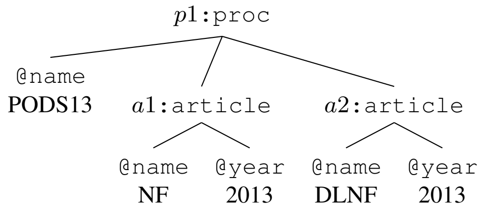
The same information is representable in different data models. For example, consider an application that manages information on conference articles. For each article we want to store the name, the conference proceedings it appeared in, and the year of its publication. The choice of the data model clearly depends on the application. For example, a relational table or database as shown in Figure 1.1a is the most traditional form of data storage. If we aim at a Web-based application an XML document, depicted as a tree in Figure 1.1b, is an option. Another possibility is to use a graph database as shown in Figure 1.1c.

Regardless of the data model, the design of a data store is not a trivial task. Database design is a research area of its own and well-studied in the relational model. One of the main questions in database design is: *"How to maintain the consistency of data?"*. This question directly comes into play when we consider consistency in the context of data dependencies, which add semantic information to data. The most prominent kinds of problems associated with data dependencies are the following [1]:

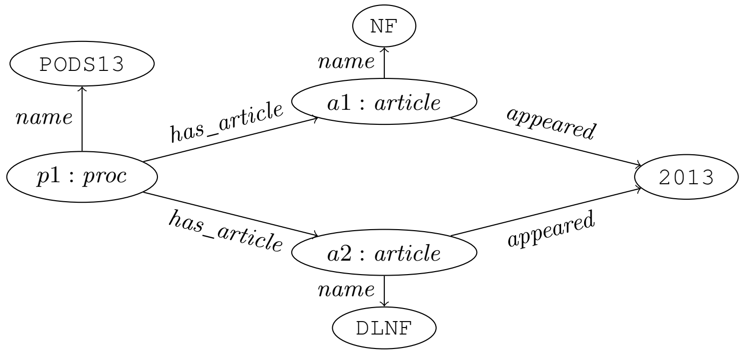
- *Incomplete Information*: If one needs to insert incomplete information into a database, this might not be possible due to a data dependency, which leads to an *insertion anomaly*. The deletion of information needed by a data dependency may lead to a *deletion anomaly*.

proc	*article*	*year*
PODS13	NF	2013
PODS13	DLNF	2013

(a) Data stored in a table.



(b) Data stored in a tree.



(c) Data stored in a graph.

Figure 1.1: Information on conference articles stored in different data models.

- *Redundancy*: Updates to redundantly stored data must be applied to all instances of it. Updating only one data instance leads to an *update anomaly*.

Figure 1.1 shows redundant information in all three data models. Articles that appear in the same conference proceedings should have the same year of publication. For example, the two articles used in Figure 1.1 appeared in the conference proceedings of “PODS13”. Both were published in 2013. In our simple example, the year is stored redundantly in each data model. Intuitively, this is due to the fact that year is modeled as a “property” of each article. Instead, it should be treated as an attribute of conference only. In the following we want to show how we can avoid and repair such redundancies.

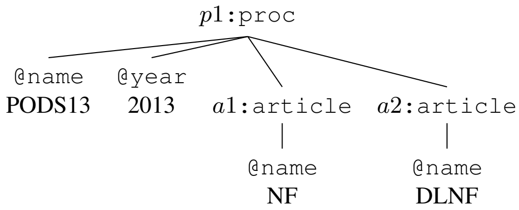
For the relational model most of the research to eliminate such problems was conducted in the 1970s [32, 33]. The *normalization process* [18] was developed to make data models as redundancy-free as possible. Such models are said to be in a particular *normal form*. Today the most widely used normal form is Boyce-Codd Normal Form (BCNF) [34].

As we have seen in Figure 1.1b, XML documents may contain redundancies as well. Arenas and Libkin analyzed redundancies in XML data. This led to the introduction of “A Normal Form for XML documents” [6], called *XML Normal Form* (XNF). They showed that XNF is a faithful extension of BCNF, using a mapping from relational data to XML documents. Under such a mapping, an XML document is in XNF if the underlying relational schema is in BCNF and vice versa.

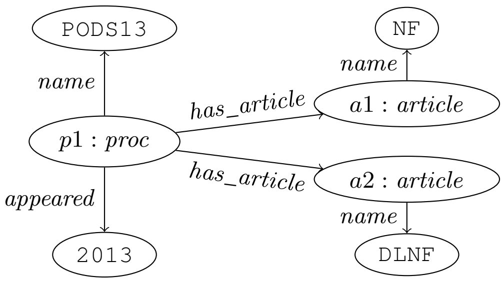
proc	*article*
PODS13	NF
PODS13	DLNF

proc	*year*
PODS13	2013

(a) Data stored in tables.



(b) Data stored in a tree.



(c) Data stored in a graph.

Figure 1.2: Information on conference articles stored in different data models.

So far, redundancies in graph databases have not yet been investigated. This thesis aims at filling this gap. Therefore, **our goal is to introduce a normal form for graph databases**. Graph databases in this normal form avoid the storage of redundant information. Similar to Arenas and Libkin, we want to justify the naturalness of such a normal form as an extension to BCNF. When we translate a relational schema into a graph database we will show that then this graph database is in our new normal form whenever the relational schema is in BCNF and vice versa.

In order to achieve this goal, we need to understand the normalization process in the relation model as well as in XML. The success of the normalization process is due to its generality. It does not aim at removing redundant data from a database, but rather it reorganizes the underlying **structure** to avoid storing redundant information. The structure for relational data is given by a relational schema [31]. A relational schema consists of tables and their columns. Several constraints, known as data dependencies, can be specified to add semantic information to the columns in a table. One type of data dependencies are functional dependencies (FDs). An FD specifies that one or more columns uniquely determine the value of another column in a relation. For example, we can specify that the data in the $proc$ column determines the data in the $year$ column. Such information leads to redundancies. We can then adapt the relational schema to make it redundancy-free. In the example, this means creating an additional table (see Figure 1.2a) that stores the information $year$ together with the $proc$ column.

XML documents can be represented as trees. The structure of these trees is determined by a Document Type Definition (DTD) [66] or some other form of XML schema (eg. [68]). Arenas and Libkin focused on DTDs [8] and introduced data dependencies over DTDs, called

XML Functional Dependencies (XFDs). XFDs are used to specify that one or more nodes in an XML tree uniquely determine another node. Therefore, XFDs are the analogue to FDs in XML documents. For example, we can specify that the $proc$ element in the tree in Figure 1.1b determines the $year$ attribute of all its $article$ child elements. This leads to redundancies and can be repaired, as depicted in Figure 1.2b, by changing the DTD, such that $year$ is an attribute of the $proc$ element rather than the $article$ element.

Redundancies in graph databases seen from a relational or XML perspective have not been investigated so far. Several models exist for graph databases. We want to focus on a particular well-suited data model, which should allow us to express constraints over graph databases. Description Logics (DLs) are an ideal choice as a constraint language for graph databases, because their models can be represented as graphs. DLs are a fragment of first-order logic. In DLs there is always a trade-off between expressivity and computational complexity. We are looking for a DL that is expressive enough to allow us to formulate FD-like constraints. For this, *DL-Lite*$_\mathcal{A}$ [24] is an ideal candidate as a formal model. *DL-Lite*$_\mathcal{A}$ is tightly related to conceptual modeling formalisms and is actually able to capture their most important features [12].

DLs and especially *DL-Lite*$_\mathcal{A}$ already allow to express constraints over the models. But, in order to establish a normal form for *DL-Lite*$_\mathcal{A}$ that resembles BCNF and XNF, we need data dependencies that are similar to FDs and XFDs. Therefore, we want to express that one or more nodes in a graph uniquely determine another node. Calvanese et al. proposed an extension to *DL-Lite*$_\mathcal{A}$, called *path-based identification constraints* [27], that allows such expressions. However, we will show that path-based identification constraints are not powerful enough to capture FDs in DLs. Therefore, we introduce *tree-based identification constraints*. With these constraints we can specify, in the example given in Figure 1.1c, that every node of type $proc$ determines the node reachable via a path that traverses a "has_article" and an "appeared" edge. Such a constraint leads to a redundancy. After moving the "appeared" edge to the $proc$ element, the graph-database is redundancy-free.

As we saw in our simple example, redundancies in graph databases can occur when a node is determined by another node, which is reachable via two or more different paths. We will generalize this idea and define a normal form for *DL-Lite*$_\mathcal{A}$, called *Description Logic Normal Form* (DLNF).

Since BCNF is the most prominent normal form in the relational model, we want to show that DLNF generalizes BCNF. As Arenas and Libkin used a mapping from relational schemas to XML, we want to translate relational schemas to *DL-Lite*$_\mathcal{A}$ knowledge bases (KB). It does not suffice to use the mapping from relational data into RDF graphs [5], because it lacks means of translating the semantic information given in the relational schema. We will extend, similar to Sequeda et al. [63], the mapping from relational data into RDF graphs and introduce the *Relational to Description Logic Direct Mapping* (R2DM). We can show for our direct mapping that any model of the translated KB corresponds to an instance of the relational schema. Our extension allows us then to show that this translated KB is in DLNF if the corresponding relational schema is in BCNF and vice versa.

Contributions. The main contributions of this thesis are the following:

- **Relational to Description Logic direct mapping**: The direct-mapping of relational data to RDF [5] lacks means of mapping the semantic information available in the relational schema. Sequeda et al. already extended this direct mapping with semantic information [63]. We will transfer their ideas to DLs and introduce in Section 4.2.2 the *relational to Description Logic direct mapping* (R2DM). The R2DM translates relational schemas into *DL-Lite*$_{\mathcal{A}}$ KBs. Then it is possible to translate models of such KBs into instances of relational schemas and vice versa. We will show that any model of such a KB can be translated into an instance of the relational schema and vice versa.

- **Tree-based identification constraints**: In order to establish a normal form that is related to BCNF, we need to express dependencies over *DL-Lite*$_{\mathcal{A}}$ KBs that resemble FDs over relational schemas. Calvanese et al. introduced path-based identification constraints [27] as such a formalism. In Section 4.3.2 we will show that these do not properly capture FDs. Therefore, we introduce *tree-based identification constraints* as a solution in Section 4.3.3. Additionally, we will show that any FD of a relational schema can be translated into a tree-based identification constraint over a *DL-Lite*$_{\mathcal{A}}$ KB and vice versa.

- **Description Logic Normal Form**: In Section 4.4 we will investigate redundancies in graph databases and establish a normal form for DLs. A *DL-Lite*$_{\mathcal{A}}$ KB in *Description Logic Normal Form* avoids redundancies. Additionally, we will compare in Section 4.5 DLNF to BCNF. We will show that whenever a relational schema is in BCNF its translated *DL-Lite*$_{\mathcal{A}}$ KB is in DLNF and vice versa.

State of the Art.

- Most of the research on the normalization of schemas in the relational model has been conducted in the 1970s and 1980s. In 1970 Codd introduced the relational model [31]. In this seminal paper he also coined the concept of normalization by giving a definition of what we today know as "First Normal Form" (1NF). Since then normal forms for relational data have been investigated. Codd developed Second and Third Normal Form (2NF and 3NF) [32, 33]. Additionally, Boyce-Codd Normal Form (BCNF) was introduced [34]. So far, all normal forms have used functional dependencies for specifying semantic information on the data. Fagin introduced Fourth and Fifth Normal Form (4NF and 5NF) that avoid redundancies when multivalued and join dependencies are used [38, 39]. Today even more normal forms exist. Among them are Domain/Key Normal Form (DKNF) [40], Sixth Normal Form (6NF) [36] and more recently, Essential Tuple Normal Form (ETNF) [35]. We will give a thorough introduction to the relational model, FDs, 3NF and BCNF in Chapter 2.

- In 1998 the World Wide Web Consortium (W3C) introduced XML as a human- and machine-readable data format for the WWW [66, 67]. A first normal form for XML was developed by Embley and Mok [37]. This normal form is more restrictive than the normal form later introduced by Arenas and Libkin [4, 6, 8]. To further investigate redundancies in relational data and XML documents Arenas and Libkin looked at these from an information-theoretic perspective [7]. Embley and Mok as well as Arenas and Libkin developed their own functional dependency language for XML documents. Additionally, both groups showed how to convert poorly designed into well-designed XML schemas. An additional language for functional dependencies in XML documents has been developed [52]. However, the authors have not considered the normalization problem with respect to the introduced dependencies. The model for XML documents together with XFDs and XNF as introduced by Arenas and Libkin will be recapitulated in Chapter 3.

- Graph databases, introduced in the late 1980s [50, 51], are an active field of research today [14, 15, 55]. They play an important role in today's applications, for example the Semantic Web [20,64]. The development of the Semantic Web started in 2001. Since then the Resource Description Framework (RDF) [44] has emerged as a standard for storage of Web data as a graph database. RDF Schema (RDFS) [22] and the Web Ontology Language (OWL) [16, 65] make it possible to attach more semantic information to RDF data. The semantics of OWL2 [65] is given by an extension of the semantics of the DL $\mathcal{SROIQ}$ [59]. In addition to the standard semantics of OWL2, the W3C defined OWL2 profiles tailored for specific purposes [58]. OWL2 QL was defined to be used with large volumes of instance data, as nowadays available in DBMS. At the core of OWL2 QL is the description logic *DL-Lite*$_{\mathcal{A}}$ [24], a member of the *DL-Lite* family [10]. This makes description Logics [13] well-suited for managing data repositories [53]. In particular, DLs are a natural language for constraints over graph databases [30]. Many extensions [27,29] to classic DLs are available that are tailored towards specific applications [25].

  Several formalisms for expressing functional dependencies have been investigated in DLs. Among them, Calvanese et al. introduced identification constraints [29] and extended them with path-based identification constraints [27]. A more restrictive form of FDs in DLs is investigated in [61]. They only consider FDs in DLs over functional paths. To the best of our knowledge there is no research on normal forms regarding these types of FDs in DLs.

Organization. This thesis is organized as follows. We will investigate redundancies and normal forms in each of the three introduced data models. Therefore, this thesis has three chapters, each of which focuses on a particular data model. Chapter 2 discusses normal forms for the relational model, chapter 3 summarizes the normal form for XML documents introduced by Arenas and Libkin [8]. In Chapter 4 we will finally present a new normal form for DLs. Each chapter has the following sections: First, we will introduce the "Preliminaries" for the particular data model. Then, we show how to map relational schemas to schemas of the particular model. Next, we investigate data dependencies. Then, we will introduce normal forms and show their correlation to BCNF. Finally, the last section of each chapter is devoted to a short summary. Chapter 5 will then review this work and give an outlook to further work in this area.

CHAPTER 2

Existing Normal Forms For Relational Data

This chapter summarizes the foundations of Database Design Theory. This summary is similarly structured as in [1, 3, 56]. Section 2.1 gives a short introduction to the relational model. Then, Section 2.2 introduces one type of data dependencies for the relational model, called functional dependencies. Finally, Section 2.3 introduces the two most prominent normal forms for the relational model, Third Normal Form and Boyce-Codd Normal Form.

2.1 Preliminaries

Codd introduced in [31] the relational model which is used by most database management systems (DBMS) today. Such a DBMS stores data in relations (or tables). Figure 2.1 shows an example relation storing information about courses of universities. A tuple (or row) of *course* stores a lecture, its type and the room together with the rooms' location.

As we have seen, each relation consists of two parts: the actual data, which varies over time, and the part considered to be fixed, the *relational schema*. A schema, denoted as $R[U]$, consists of a *schema name* (R) and a set of *attributes* $U = \{A_1, \ldots, A_n\}$. The values stored in each attribute $A \in U$ are of a particular *domain* denoted as $Dom(A)$. We assume the domains to be infinite. The schema shown in Figure 2.1 is $course[U]$, where $U = \{lecture, type, room, building\}$ and $Dom(lecture) = \Sigma^*$, i.e. the set of all strings. A *tuple* is a function with domain U that assigns to each attribute of a relation a value of the domain. An *instance* I of a relation consists of a set of tuples. For example, the instance depicted in Figure 2.1 is $I(course) = \{t_1, t_2, t_3\}$, where $t_1(lecture) =$ "Algebra I", and so on. Let $X \subseteq U$, we denote with $t[X]$ the tuple restricted to the attributes in X. Given a set of tuples T, e.g. $I(course)$, we denote with $T[X]$, the set of tuples restricted to the attributes in X. A *database schema* $\boldsymbol{S}$ consists of several relational schemas, i.e. $\boldsymbol{S} = \{R_1[U_1], \ldots R_n[U_n]\}$. Additionally, constraints might

$lecture$	$type$	$room$	$building$
Algebra I	VO	HS1	Main
Algebra I	UE	SEM1	Dep
Economics I	UE	SEM1	Dep

Figure 2.1: Relation $course$

be imposed over a relational schema. Such constraints restrict the possible relational schema instances. For example, the instance in Figure 2.1 depicts that one room is associated to a building. In order to express such constraints, we need to add *data dependencies* to a relational schema. Different classes of data dependencies will be considered in Section 2.2. A relational schema $R[U]$ together with some set of dependencies Σ, denoted by $(R[U], \Sigma)$, is, for the sake of simplicity, also called relational schema [3]. We denote by $Inst\ (R\ [U])$ the set of all possible instances of the relational schema $R\ [U]$.

2.2 Data Dependencies

Data dependencies impose constraints over an instance of a relational schema. Let us first introduce two concepts common to all classes of data dependencies, *dependency implication* and *dependency inference*. Let φ denote a dependency and Σ a set of dependencies. A set of dependencies Σ implies a dependency φ, denoted by $\Sigma \models \varphi$, if for every database instance I that satisfies all constraints in Σ, it is the case that I satisfies φ. The set of all dependencies implied by Σ is denoted by Σ^+. Dependency implication can be decided using different methods. On the one hand, there are algorithms, on the other hand we can try to construct a proof for the implication of a FD by using an inference system. Such an inference system consists of a set of inference rules $\mathcal{I}$. Let $\mathcal{C}$ be a class of dependencies, e.g. the class of Functional Dependencies. We say a set of rules $\mathcal{I}$ is *complete* for a class of dependencies $\mathcal{C}$, if for every set $\Sigma \cup \{\varphi\}$, if $\Sigma \models \varphi$, then $\Sigma \vdash_{\mathcal{I}} \varphi$, i.e. φ is provable from Σ using $\mathcal{I}$. Furthermore, $\mathcal{I}$ is *sound* for $\mathcal{C}$ if $\Sigma \vdash_{\mathcal{I}} \varphi$ implies $\Sigma \models \varphi$. The three most important classes of dependencies are Functional Dependencies, Multi-Valued Dependencies and Join Dependencies. Boyce-Codd Normal Form uses Functional Dependencies. Since in this work we primarily focus on BCNF, we only introduce Functional Dependencies here. We exhibit the implication problem for Functional Dependencies and establish a set of sound and complete inference rules.

2.2.1 Functional Dependencies

Functional Dependencies (FDs) are the most important class of dependencies in schema design. Key dependencies, a special type of FDs, are supported by most DBMS today. Let U be a set of attributes, and let $X, Y \subseteq U$. A *Functional Dependency* over U is an expression of the form $X \to Y$. If $Y = U$ then $X \to Y$ is called a *key dependency*. An FD is called *trivial* if $Y \subseteq X$. An instance I of $R[U]$ satisfies an FD $X \to Y$, denoted by $I \models X \to Y$, if for every pair of tuples t_1, t_2 in I, whenever $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$. Intuitively, an FD says that if two tuples agree on the values of X they must also agree on the values of Y.

Now consider the relation $course$ in Figure 2.1. We already noted that each room is associated to a building. Therefore, $room \to building$ is an FD over the relation $course$. Additionally, a lecture together with its type determines the room they can be held in. Also, rooms can only serve lectures of a particular type. Therefore the set of FDs Σ_{course} over the relation $course$ is:

$$room \to building \tag{2.1}$$
$$lecture, type \to room \tag{2.2}$$
$$room \to type \tag{2.3}$$

We have now specified the relational schema $(course[U], \Sigma_{course})$, with $U = \{lecture, type, room, building\}$. Notice, that the relation in Figure 2.1 is a valid instance of this schema.

2.2.1.1 Implication of FDs

The implication of a particular FD φ by a set of FDs Σ can be determined by computing the closure of a set of attributes X. Let $X \subseteq U$ be a set of attributes. The *closure* of X, denoted by X^+, are all attributes implied by X, i.e. $X^+ = \{A \mid \Sigma \models X \to A \wedge A \in U\}$. A naive algorithm computes X^+ in $O(n^2)$, where n is the length of Σ and X. Algorithm 2.1 developed by Beeri and Bernstein [17,21] computes X^+ in linear time. We can now check if an FD $\varphi = X \to Y$ is implied by a set of FDs Σ, since $\Sigma \models X \to Y$ if and only if $Y \subseteq X^+$. Therefore, implication of FDs can be decided in linear time.

As an example, consider the FDs Σ_{course} and the FD $\varphi_{course} = lecture, type \to building$. We want to check if $\Sigma_{course} \models \varphi_{course}$. First, we use Algorithm 2.1 to compute $\{lecture, type\}^+ = \{lecture, type, room, building\}$. Then φ_{course} is implied by the FD Σ_{course}, since $building \in \{lecture, type\}^+$.

2.2.1.2 Axiomatization for FDs

An alternative to the implication problem is to provide a proof for an FD. Such a proof is constructed using inference rules. Armstrong introduced in [9] the following sound and complete set of inference rules, also called Armstrong Axioms:

FD1 (Reflexivity): If $Y \subseteq X$, then $X \to Y$ (2.4)

FD2 (Augmentation): If $X \to Y$, then $XZ \to YZ$ (2.5)

FD3 (Transitivity): If $X \to Y$ and $Y \to Z$, then $X \to Z$ (2.6)

For example, φ_{course} can be inferred by applying transitivity (FD3) to $lecture, type \to room$ and $room \to building$.

input : A set U of attributes, a set Σ of functional dependencies over U, and a set $X \subseteq U$.
output: The closure X^+ of X with respect to Σ

unmark all members of X;
foreach $\varphi = Y \to Z \in \Sigma$ **do**
  $count(\varphi) \leftarrow |Y|$;
  foreach $A \in Y$ **do**
    add φ to the list $L(A)$;
  end
end
$CL \leftarrow X$;
while *CL contains an unmarked element* A **do**
  mark A;
  foreach $\varphi \in L(A)$ **do**
    $count(\varphi) \leftarrow count(\varphi) - 1$;
    if $count(\varphi) = 0$ **then**
      let $\varphi = Y \to Z$;
      $CL \leftarrow CL \cup Z$;
    end
  end
end

Algorithm 2.1: Linear time algorithm to compute the closure X^+ of a set of attributes X (from [56])

2.3 Normal Forms

Normal forms are one of database theory most important contributions to schema design. The goal of normal forms is to formulate criteria for "good" relational schemas. Intuitively, such "good" schemas should help to avoid redundant information, and update, insertion and deletion anomalies.

Before we define normal forms, we need some auxiliary definitions. Let $(R[U], \Sigma)$ be a relational schema with the functional dependencies Σ. We call $X \subseteq U$ a *superkey* if $\Sigma \models X \to U$. A *key* is a minimal superkey. The attributes A in X, where X is a key of R, are called *key attributes* [1].

We only study Third Normal Form (3NF) and Boyce-Codd Normal Form (BCNF) here. For completeness, First Normal Form (1NF) states that attributes in relations are atomic [56], which is an assumption already made by the relational model. Second Normal Form (2NF) demands that all non-key attributes should depend on the whole key of a relation [56]. All normal forms require the relation to be in the weaker normal form, i.e. 3NF requires that the relation is in 2NF and 1NF.

2.3.1 Third Normal Form

Third Normal Form was first proposed in [32, 33] in order to avoid update anomalies.

Definition 2.1. (Third Normal Form [1, 69]) Let $(R\,[U]\,, \Sigma)$ be a relational schema, where Σ is a set of functional dependencies. $(R\,[U]\,, \Sigma)$ is in *third normal form (3NF)* if whenever $X \rightarrow A$ is a nontrivial FD implied by Σ, then X is a superkey or A is a key attribute. $\triangleleft$

In other words, Definition 2.1 states that whenever an attribute A is functionally dependent on another set of attributes X, then X is a superkey or A is part of the key of R. We will now illustrate 3NF with the following example.

Example 2.1. Consider the relational schema $(course[U], \Sigma_{courses})$ introduced in Section 2.1 and extended with FDs in Section 2.2. Now consider the FD $room \rightarrow building$. Neither $room$ is a superkey, since $room \rightarrow lecture$ is not a valid FD in the relational schema, nor $building$ is part of the key of R, which is $lecture, type, room$. Therefore, this schema is not in 3NF. We can split the relation $course$ into a relation $course$ with attributes $lecture$, $type$, and $room$ and into a relation $rooms$ with attributes $room$ and $building$. Notice, that this new set of relations $course$ and $room$ is in 3NF. $\triangleleft$

2.3.2 Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) was introduced in [34] and can be summarized with "Do Not Represent the Same Fact Twice" [1], which eliminates redundancies and update anomalies. BCNF is defined as follows:

Definition 2.2. (Boyce-Codd Normal Form [1] Let $(R\,[U]\,, \Sigma)$ be a relational schema, where Σ is a set of functional dependencies. $(R[U], \Sigma)$ is in *Boyce-Codd normal form* (BCNF) if $\Sigma \models X \rightarrow U$ whenever $X \rightarrow Y$ is a nontrivial FD implied by Σ. A database schema $(\mathbf{R}, \Sigma)$ is in BCNF if each of its relation schemas is. $\triangleleft$

Put differently, a schema is in BCNF if for every nontrivial functional dependency $X \rightarrow A \in \Sigma^+$, X is a superkey [56]. Thus, compared to 3NF, BCNF drops the condition that A might be part of the key. The algorithm for testing BCNF works as follows: For every FD $X \rightarrow Y \in \Sigma$, we compute X^+ using Algorithm 2.1. If $X^+ = U$ then we continue with the next FD. Since Algorithm 2.1 runs in linear time, we can decide if a relational schema is in BCNF in quadratic time. Example 2.1 shows a relational schema that is in 3NF. Next, we show that this relational schema is not in BCNF:

Example 2.2. Let $course(lecture, type, room)$ be the relational schema obtained after decomposition into 3NF. The FDs for the relational schema $course$ are
$(lecture, type \rightarrow room)$ and $(room \rightarrow type)$. This schema is not in BCNF since $room \rightarrow lecture$ is not implied by the above FDs. An instance of this schema is presented in Figure 2.2. Let us explain at this example the BCNF intuition "Do Not Represent the Same Fact Twice". We extract the columns used in the FD that leads to the violation of BCNF. Those are $room$ and $type$. The instance in Figure 2.2 restricted to $room$ and $type$ consists of three rows, where two

$lecture$	$type$	$room$
Algebra I	VO	HS1
Algebra I	UE	SEM1
Economics I	UE	SEM1

Figure 2.2: Relation $course$ in 3NF

rows appear twice (those are $(\mathit{UE}, \mathit{SEM1})$). Since the instance in Figure 2.2 is a valid instance we have stored *twice* the information that in the room SEM1 courses of the type UE can be taught. $\triangleleft$

Relational schemas that are not in BCNF can be repaired. We call such a repair algorithm a *decomposition*. The basic idea of a BCNF decomposition algorithm is, to find a FD $X \rightarrow Y$ which leads to a BCNF violation. Notice that Y have to be all attributes implied by X. We then create a new relation for the attributes X and Y and remove the attributes Y from the original relation. Therefore, we have repaired the BCNF violation, that has originated from the FD $X \rightarrow Y$. Such a decomposition should preserve data and dependencies. The BCNF decomposition algorithm preserves the data, but, unfortunately, is not dependency preserving (see Theorem 11.2.8 of [1]). The next example illustrates the loss of a dependency due to the BCNF decomposition algorithm.

Example 2.3. Consider the relational schema and instance given in Figure 2.2 of Example 2.2. Since the FD $room \rightarrow type$ leads to a BCNF violation, we create a new relation $rooms$ with the attributes $room$ and $type$. We remove the attribute $type$ from the relation $course$. Additionally, we need to drop the FD $lecture, type \rightarrow room$, since $type$ does not belong to the relation $course$ anymore. Therefore, the algorithm does not preserve the FD $lecture, type \rightarrow room$. The result is the database schema in BCNF consisting of the relations:

$$(course(lecture, room), \emptyset) \qquad (rooms(room, type), \{room \rightarrow type\})$$

The resulting database instance with the same information as in Figure 2.2 is given in Figure 2.3. Unfortunately, there is no BCNF decomposition of $course$ that preserves dependencies. $\triangleleft$

$course$

$lecture$	$room$
Algebra I	HS1
Algebra I	SEM1
Economics I	SEM1

$rooms$

$room$	$type$
HS1	VO
SEM1	UE

Figure 2.3: Relation $course$ and $rooms$ in BCNF

2.4 Summary

In this chapter we have introduced the relational model as a formal model of relational databases. Then, we established functional dependencies as a formalism for data dependencies over a relational schema. We have investigated the implication problem for FDs. The presented algorithm decides the implication problem for FDs in linear time. Additionally, we can use Armstrong Axioms to prove the implication of FDs.

Then, we introduced two different normal forms for the relational model. A relational schema R is in Third Normal Form if for every attribute A that is functionally dependent on some other attributes X, then X is a superkey or A is a key attribute. This normal form avoids update anomalies. Boyce-Codd Normal Form eliminates redundancies and update anomalies. BCNF drops the condition that A might be a key attribute. Therefore, BCNF is more restrictive than 3NF. If a relational schema is in BCNF can be checked in quadratic time. Clearly, every schema that is in BCNF is also in 3NF. At the end of this chapter we have showed that we can repair relational schemas that are not in BCNF, such that the resulting relational schema is in BCNF.

CHAPTER 3

Existing Normal Form For XML Data

XML documents are increasingly used in today's applications for storing and exchanging data. The data in those XML documents is retrieved, updated and inserted. XML documents have their own structure, determined by a Document Type Definition (DTD) [66] or some other form of XML schema (eg. [68]). Arenas and Libkin looked for an analogon to BCNF in the XML context [8]. As a summary they answered the following questions:

(1) What is a redundancy and an update anomaly in XML?

(2) What do functional dependencies in XML look like?

(3) What are "bad" functional dependencies?

(4) Is there an algorithm to convert an arbitrary DTD into one without "bad" functional dependencies?

This chapter summarizes the answers to questions (1)-(3), given in the publications of Arenas and Libkin [4, 6–8]. Section 3.1 introduces the XML document and schema model. We then show how to map relational data to XML documents and DTDs in Section 3.2. Section 3.3 defines XML functional dependencies (XFD) and finally, Section 3.4 introduces XML Normal Form.

XML Documents. An XML document is hierarchically structured and built of elements. An element contains a string or a sequence of further elements. Elements start with a *start-tag*, e.g. `<course>`, and end with an *end-tag*, e.g. `</course>`. Elements might also include attributes given in the start-tag, e.g. the attribute `type` in the element `course` (`<course type='VO'>`). The top element of a document is called the *document* or *root* element. An XML document with the same information as in Figure 2.1 is given in the next example.

```
<courses>
  <course type="VO">
    <lecture>Algebra I</lecture>
    <room building="Main">HS1</room>
  </course>
  <course type="UE">
    <lecture>Algebra I</lecture>
    <room building="Dep">SEM1</room>
  </course>
  <course type="UE">
    <lecture>Economics</lecture>
    <room building="Dep">SEM1</room>
  </course>
</courses>
```

Figure 3.1: An XML document with the same information as in Figure 2.1.

Example 3.1. The root element of the document in Figure 3.1 is `courses`, which stores several `course` elements. Each `course` element has as attribute a course `type`. The name of the course is stored in a `lecture` element, and its location in a `room` element. The `room` element has a `building` attribute. $\triangleleft$

Document Type Definitions (DTDs). A schema of an XML document defines the allowed trees. This structure is defined using a schema language, which is most often given by a DTD (defined in [66]) or an XML Schema (XSD) [68]. In the following we focus on DTDs as a schema language. The next example gives a possible schema to the XML document in Figure 3.1

Example 3.2. The DTD given in Figure 3.2 allows for XML documents with the root element `courses`. Each `course` element has zero or more `course` child elements. Each `course` element has a child of type `lecture` and `room`. Additionally, the `course` element has a required attribute `type`. The `lecture` and `room` elements contain strings (`#PCDATA`). The `room` element has a `building` attribute. $\triangleleft$

3.1 Preliminaries

In this section we formally introduce XML documents. For XML documents and DTDs we use the same formal model as in [3] originally introduced by Fan and Libkin [42, 43]. We have the following disjoint sets: El representing element names, Att attribute names, Str possible values of string-valued attributes, and $Vert$ node identifiers. We assume that all attribute names Att start with the symbol @ (an no others are starting with @). The symbols `S` and $\perp$ are not part of the previous sets. An XML document can be represented as a tree, formalized as follows. Notice that we do not allow mixed content in XML trees.

```
<!DOCTYPE courses [
  <!ELEMENT courses (course*)>
  <!ELEMENT course (lecture, room)>
  <!ATTLIST course
        type CDATA #REQUIRED>
  <!ELEMENT lecture (#PCDATA)>
  <!ELEMENT room (#PCDATA)>
  <!ATTLIST room
        building CDATA #REQUIRED>
]>
```

Figure 3.2: A DTD representing courses

Definition 3.1. (XML tree T [8]) An XML tree T is defined to be a tree $(V, lab, ele, att, root)$, where

- $V \subseteq \mathit{Vert}$ is a finite set of *vertices (nodes)*.
- $lab \ldots V \to \mathit{El}$, is a function that assigns element types to vertices.
- $ele \ldots V \to \mathit{Str} \cup V^*$, is a function that assigns to vertices its child vertices, which is either an ordered set of vertices or a string.
- $att \ldots$ a partial function $V \times \mathit{Att} \to \mathit{Str}$, such that for each $v \in V$, the set $\{@l \in \mathit{Att} \mid att(v, @l)$ is defined $\}$ is finite.
- $root \in V$ is called the *root* of T

The parent-child edge relation on V, $\{(v_1, v_2) \in V \times V \mid v_2$ occurs in $ele\,(v_1)\}$, is required to form a rooted tree. $\triangleleft$

Example 3.3. The XML document in Figure 3.1 is represented by the XML tree in Figure 3.3. This tree contains a set of nodes $V = \{v_i \mid i \in [0, 9]\}$. The nodes are of the following element types:

$$lab\,(v_0) = \texttt{courses} \qquad lab\,(v_1) = \texttt{course} \qquad lab\,(v_2) = \texttt{lecture}$$
$$lab\,(v_3) = \texttt{room} \qquad lab\,(v_4) = \texttt{course} \qquad lab\,(v_5) = \texttt{lecture}$$
$$lab\,(v_6) = \texttt{room} \qquad lab\,(v_7) = \texttt{course} \qquad lab\,(v_8) = \texttt{lecture}$$
$$lab\,(v_9) = \texttt{room}.$$

ele assigns to all nodes its children:

$$ele\,(v_0) = [v_1, v_4, v_7] \qquad ele\,(v_1) = [v_2, v_3] \qquad ele\,(v_2) = \text{“Algebra I”}$$
$$ele\,(v_3) = \text{“HS1”} \qquad ele\,(v_4) = [v_5, v_6] \qquad ele\,(v_5) = \text{“Algebra I”}$$
$$ele\,(v_6) = \text{“SEM1”} \qquad ele\,(v_7) = [v_8, v_9] \qquad ele\,(v_8) = \text{“Economics”}$$
$$ele\,(v_9) = \text{“SEM1”}.$$

The attributes in T are the following:

$$\begin{array}{ll} att\,(v_1, \texttt{@type}) = \text{“VO”} & att\,(v_3, \texttt{@building}) = \text{“Main”} \\ att\,(v_4, \texttt{@type}) = \text{“UE”} & att\,(v_6, \texttt{@building}) = \text{“SEM1”} \\ att\,(v_7, \texttt{@type}) = \text{“UE”} & att\,(v_9, \texttt{@building}) = \text{“SEM1”}. \end{array}$$

Moreover, the $root$ of the tree T is v_0. ◁

Let T_1 and T_2 be two XML trees. Then T_1 is subsumed by T_2, denoted as $T_1 \preceq T_2$, if T_2 contains T_1 as a subtree (up to reordering of child nodes). Let T be an XML tree and let $w_1. \cdots .w_n$ be a string, with $w_1, \ldots, w_{n-1} \in El$ and $w_n \in El \cup Att \cup \{\texttt{S}\}$.

Definition 3.2. (Paths in T). We call $w_1 \cdots w_n$ a path in T if there exists vertices $v_1, \ldots, v_n$, such that

- $v_1 = root$ and $lab(v_1) = w_1$
- v_{i+1} is a child of v_i and $lab(v_{i+1}) = w_{i+1}$, for each $i \in [1, n-2]$
- If $w_n \in El$, then v_n is a child of v_{n-1} and $lab(v_n) = w_n$
- If $w_n = @l$, then $att(v_{n-1}, @l)$ is defined
- If $w_n = \texttt{S}$, then v_{n-1} has a child in Str

$paths(T)$ denotes the set of all paths in T. ◁

For example, `courses.course.@type` and `courses.course.room.`S are paths in the XML tree of Figure 3.3. Next, we define the formal model for DTDs.

Definition 3.3. (DTD [8]) A Document Type Definition is defined to be $D = (E, A, P, R, r)$, where:

- $E \subseteq El$ ... finite set of *element types*
- $A \subseteq Att$ ... finite set of *attributes*
- P ... a mapping from E to *element type definitions*, i.e. let $\tau \in E$, then $P(\tau) = \texttt{S}$ or a regular expression α defining the child elements of $\tau \in E$, where

  $$\alpha := \epsilon \mid \tau' \mid \alpha|\alpha \mid \alpha, \alpha \mid \alpha^*.$$

  The symbol ϵ denotes the empty sequence; τ' is an element of E; “|” denotes union, “,” concatenation and “*” the Kleene closure.
- R ... a mapping from E to the powerset of A, i.e. attributes that are available at an element $\tau \in E$.
- $r \in E$ ... the *element type of the root* ◁

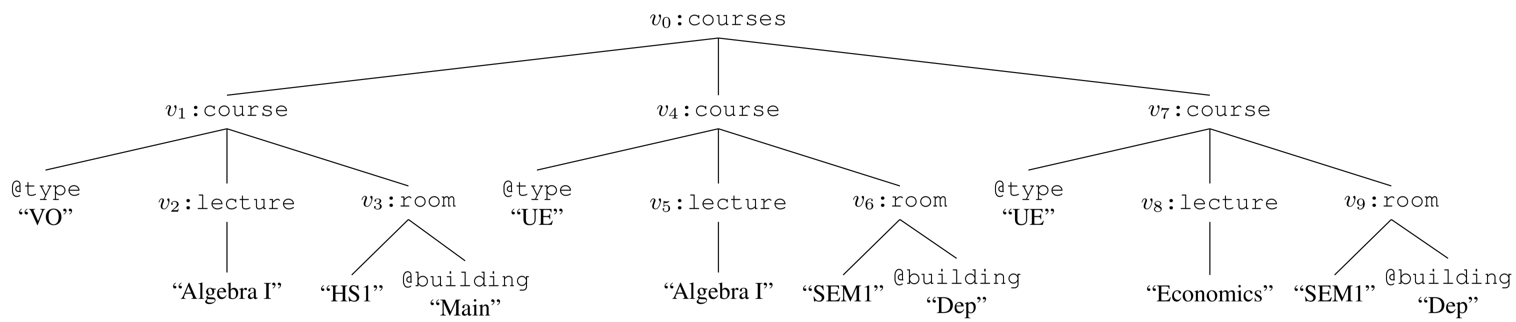


Figure 3.3: The tree representation of the XML document in Figure 3.1.

The next example shows the translation of the DTD in Figure 3.2 into the definition given above. Notice that the symbol `S` represents the element type declaration `#PCDATA`.

Example 3.4. The DTD in Figure 3.2 describes the same information as given in Figure 2.1. By Definition 3.3, we represent the DTD given in Figure 3.2 as $D_c = (E_c, A_c, P_c, R_c, courses)$, where

- $E_c = \{courses, course, lecture, room\}$;
- $A_c = \{@type, @building\}$;
- the mapping P_c is defined as:
  - $P_c(courses) = course^*$,
  - $P_c(course) = lecture, room$,
  - $P_c(lecture) = \mathtt{S}$,
  - $P_c(room) = \mathtt{S}$;
- the mapping R_c is defined as:
  - $R_c(courses) = \emptyset$,
  - $R_c(course) = \{@type\}$,
  - $R_c(lecture) = \emptyset$,
  - $R_c(room) = \{@building\}$. ◁

In order to navigate through an XML tree we introduce the notion of paths in a DTD D.

Definition 3.4. (Paths in D) A path w is a sequence of elements or attributes, i.e.

$$w = w_1 \dots w_n.$$

A path is in a DTD D if

- $w_1 = r$,
- w_i is in the alphabet of $P(w_{i-1})$, for each $i \in [2, n-1]$, and
- w_n is in the alphabet of $P(w_{n-1})$ or $w_n = @l$ for some $@l \in R(w_{n-1})$. ◁

The length of a path $w = w_1 \dots w_n$ is denoted by $length(w) = n$. We denote by $paths(D)$ the set of all paths in a DTD D, and with $EPaths(D)$ the set of all paths that end with an element type, rather than an attribute, i.e. $EPaths(D) = \{p \in paths(D) \mid last(p) \in E\}$, where $last(p) = w_n$. If the set $paths(D)$ is infinite, then we call the DTD D recursive.

Example 3.5. Let D_c be the DTD introduced in Example 3.4. Then, the set of paths in D_c is

$$\begin{aligned} paths\,(D_c) = \{&courses,\\ &courses.course,\\ &courses.course.@type,\\ &courses.course.lecture,\\ &courses.course.lecture.\mathtt{S},\\ &courses.course.room,\\ &courses.course.room.\mathtt{S},\\ &courses.course.room.@building\}, \end{aligned}$$

and the set

$$\begin{aligned} EPaths\,(D_c) = \{&courses,\\ &courses.course,\\ &courses.course.lecture,\\ &courses.course.room\}. \end{aligned}$$

$\triangleleft$

Since a DTD determines the allowed XML tree, we need to define these. An XML tree T conforms to a DTD D, denoted by $T \vDash D$, if the following holds:

Definition 3.5. ($T \vDash D$ [8]) Given a DTD $D = (E, A, P, R, r)$ and an XML tree $T = (V, lab, ele, att, root)$, we say that T conforms to D ($T \vDash D$) if

- lab is a mapping from V to E.
- For each $v \in V$,
  - if $ele(v) = s$, where $s \in Str$, then $P(lab(v)) = \mathtt{S}$.
  - if $ele(v) = [v_1, \ldots, v_n]$, then the string $lab\,(v_1) \cdots lab\,(v_n)$ must be in the regular language defined by $P(lab(v))$.
- att is a partial function from $V \times A$ to Str, s.t. for any $v \in V$ and $@l \in A$, $att\,(v, @l)$ is defined iff $@l \in R(lab(v))$.
- $lab\,(root) = r$.

$\triangleleft$

For example, the XML tree shown in Figure 3.3 conforms to the DTD shown in Figure 3.2. Additionally, we say that T *is compatible with* D, denoted by $T \triangleleft D$, iff $paths(T) \subseteq paths(D)$. Notice that a tree T is compatible with a DTD D if T conforms to D but not vice-versa.

```
<!DOCTYPE db [
  <!ELEMENT db (course*)>
  <!ELEMENT course EMPTY>
  <!ATTLIST course
       lecture CDATA #REQUIRED
       type CDATA #REQUIRED
       room CDATA #REQUIRED
       building CDATA #REQUIRED>
]>
```

Figure 3.4: A DTD mapped from the relational schema $course$

3.2 A Direct-Mapping From Relational Data To XML Documents

Relational data is easily mapped into XML documents. In this section we introduce the direct-mapping of relational data into XML documents as defined in [8]. Let $G(A_1, \ldots, A_n)$ be a relational schema. The direct-mapping into an XML representation outputs a DTD. Such a DTD $D_G = (E, A, P, R, db)$ is defined as follows:

- $E = \{db, G\}$.
- $A = \{@A_1, \ldots, @A_n\}$.
- $P(db) = G^*$ and $P(G) = \epsilon$.
- $R(db) = \emptyset$ and $R(G) = \{@A_1, \ldots, @A_n\}$.

Notice, that this DTD allows for duplicate representation of tuples (two elements of type G with the same attributes). This is inconsistent with the set semantics of the relational model. After we have introduced data dependencies for DTDs, we will extend this mapping to avoid this problem. Additionally, we also add a translation of FDs to data dependencies in XML. The next example shows the DTD D_{course} translated from the relational schema introduced with Figure 2.1.

Example 3.6. The relational schema $course(lecture, type, room, building)$ is mapped into the DTD listed in Figure 3.4. Notice that this directly-mapped DTD is different from the DTD given in Figure 3.2. $\triangleleft$

3.3 Data Dependencies

In this section we will summarize data dependencies for XML documents, called XML Functional Dependencies (XFDs), introduced in [8]. First, we need the notion of tree tuples, which gives us a natural representation of XML trees as sets of tuples. This allows us to find a natural definition for FDs in XML documents.

3.3.1 Tree Tuples

A tuple in relational databases is a total mapping from the set of attributes to domain values [1]. Tree tuples should extend the notion of relational tuples. Therefore, the function t is called a tree tuple in a DTD D, if it assigns to each path in D a value in $Vert \cup Str \cup \{\perp\}$. Each path in D occurs at most once in t.

Definition 3.6. (Tree Tuples [8]) Let $D = (E, A, P, R, r)$ be a DTD. A *tree tuple* t in D is a function from $paths\,(D)$ to $Vert \cup Str \cup \{\perp\}$, such that:

- For $p \in EPaths(D)$, $t(p) \in Vert \cup \{\perp\}$, and $t(r) \neq \perp$.
- For $p \in paths(D) - EPaths(D)$, $t(p) \in Str \cup \{\perp\}$.
- If $t(p_1) = t(p_2)$ and $t(p_1) \in Vert$, then $p_1 = p_2$.
- If $t(p_1) = \perp$ and p_1 is a prefix of p_2, then $t(p_2) = \perp$.
- $\{p \in paths(D) \mid t(p) \neq \perp\}$ is finite. $\triangleleft$

$\mathcal{T}(D)$ denotes the set of all tree tuples in D. For a tree tuple t and a path p, we write $t.p$ for $t(p)$. Also note that no path p occurs twice in a tree tuple, i.e. a tree tuple assigns to every path in $paths(D)$ exactly one value.

Example 3.7. Suppose that D is the DTD shown in Example 3.2. Then a tree tuple in D assigns values (taken from the XML tree in Figure 3.1) to each path in $paths(D)$ (listed in Example 3.5):

$$
\begin{aligned}
&t(courses) = v_0\\
&t(courses.course) = v_1\\
&t(courses.course.@type) = \text{“VO”}\\
&t(courses.course.lecture) = v_2\\
&t(courses.course.lecture.\mathtt{S}) = \text{“Algebra I”}\\
&t(courses.course.room) = v_3\\
&t(courses.course.room.\mathtt{S}) = \text{“HS1”}\\
&t(courses.course.room.@building) = \text{“Main”}
\end{aligned}
\qquad \triangleleft
$$

Notice that we only assign to finitely many paths a value different from $\perp$. Thus, even for recursive DTDs, where $paths(D)$ is infinite, we can represent the non-null values of tree tuples as XML trees as follows:

Definition 3.7. ($tree_D$ [8]) Let $D = (E, A, P, R, r)$ be a DTD and let t be a tree tuple, where $t \in \mathcal{T}(D)$. The function $tree_D(t)$ outputs an XML tree $(V, lab, ele, att, root)$ as follows: Let $root = t.r$ be the root of this XML tree and

- $V = \{v \in Vert \mid \exists p \in paths(D)$ such that $v = t.p\}$.

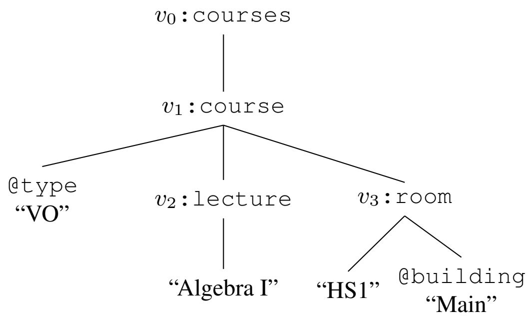


Figure 3.5: The XML tree $tree_D(t)$.

- If $v = t.p$ and $v \in V$, then
  - $lab(v) = last(p)$, and
  - $ele(v)$ is defined to be the list containing $\{t.p' \mid t.p' \neq \bot \text{ and } p' = p.\tau, \tau \in E, \text{ or } p' = p.\mathtt{S}\}$, and, since an XML tree must be ordered, this list is ordered lexicographically.
- If $v = t.p$, $@l \in A$ and $t.p.@l \neq \bot$, then $att(v, @l) = t.p.@l$. ◁

Example 3.8. Let D be the DTD from Example 3.2, and let t be the tree tuple from Example 3.7. Then, $tree_D(t)$ outputs the XML tree shown in Figure 3.5. ◁

The tree in Figure 3.5 conforms to the DTD D, which is not necessarily the case in general, since, for example, tree tuples disregard the ordering of child elements. But, by the definition of tree tuples, if $t \in \mathcal{T}(D)$ then the XML tree $tree_D(t)$ is compatible with D, i.e. $tree_D(t) \lhd D$. It is possible to capture the total amount of information in an XML tree with tree tuples. For this, we select only those tree tuples that contain the maximal amount of information. The notion of the maximal amount of information in tree tuples is defined via an ordering (denoted as $\sqsubseteq$) on tuples. Let t_1 and t_2 be two tree tuples, then $t_1 \sqsubseteq t_2$ if whenever $t_1.p$ is defined, then so is $t_2.p$, and $t_1.p \neq \bot$ implies $t_1.p = t_2.p$ [8]. We now can define the set of tree tuples of an XML tree T.

Definition 3.8. ($tuples_D$ [8]) Given a DTD D and an XML tree T such that $T \lhd D$, $tuples_D(T)$ is defined to be the set of maximal, with respect to $\sqsubseteq$, tree tuples t such that $tree_D(t)$ is subsumed by T; that is:

$$\max_{\sqsubseteq}\{t \in \mathcal{T}(D) \mid tree_D(t) \preceq T\}.$$ ◁

Example 3.9. In Example 3.7 one tree tuple for the DTD in Example 3.2 was given. The set of tree tuples $tuples_D(T)$ calculated from the tree in Figure 3.1 is:

$$\begin{aligned}\{&(v_0, v_1, \text{"VO"}, v_2, \text{"Algebra I"}, v_3, \text{"HS1"}, \text{"Main"}),\\ &(v_0, v_4, \text{"UE"}, v_5, \text{"Algebra I"}, v_6, \text{"SEM1"}, \text{"Dep"}),\\ &(v_0, v_7, \text{"UE"}, v_8, \text{"Economics"}, v_9, \text{"SEM1"}, \text{"Dep"})\}\end{aligned}$$ ◁

Notice that the tree tuples in Example 3.9 resemble the tuples given in the relational table shown in Figure 2.1. This gives evidence that tree tuples are a natural extension of tuples for XML documents. With the direct-mapping of relational data to XML documents introduced in Section 3.2 we can establish a one-to-one correspondence between the tuples in an instance I and the tree tuples of the XML tree T_I translated from I [3]. With the notion of tree tuples, it is now possible to define functional dependencies for XML.

3.3.2 XML Functional Dependencies

In this section we will define FDs for XML documents. Additionally, we inspect the implication problem of such FDs. We also show that FDs for XML documents are not axiomatizable. Last, we give a translation of relational FDs into XML FDs.

For a DTD D, an *XML functional dependency* (XFD) over D is an expression of the form $S_1 \to S_2$, where S_1, S_2 are finite nonempty subsets of $paths(D)$. The set of all FDs over D is denoted by $\mathcal{F}D(D)$. Let $S \subseteq paths(D)$, and let $t, t' \in \mathcal{T}(D)$, then $t.S = t'.S$ means $t.p = t'.p$ for all $p \in S$.

Let D be a DTD and let T be an XML tree such that $T \lhd D$. Then, T satisfies $S_1 \to S_2$, denoted as $T \models S_1 \to S_2$, if for every t_1, $t_2 \in tuples_D(T)$, $t_1.S_1 = t_2.S_1$ and $t_1.S_1 \neq \bot$ imply $t_1.S_2 = t_2.S_2$.

Example 3.10. Let D be the DTD introduced in Example 3.2. We now want to state XFDs that capture the semantic information of the FDs Σ_{course} introduced in Section 2.2.1:

- Equation 2.1 - Each room is associated to a building:

$$\texttt{courses.course.room.S} \to \texttt{courses.course.room.@building} \tag{3.1}$$

- Equation 2.2 - A lecture together with its type determine the room they can be held in:

$$\{\texttt{courses.course.lecture.S}, \texttt{courses.course.type.S}\} \to \texttt{courses.course.room.S} \tag{3.2}$$

- Equation 2.3 - Rooms can only serve lectures of a particular type:

$$\texttt{courses.course.room.S} \to \texttt{courses.course.type.S} \tag{3.3}$$

Notice that the XML tree in Figure 3.1 satisfies all three XFDs. An XML tree $T_{\not\models}$ that violates the XFD listed in Equation 3.1 is shown in Figure 3.6. This is because of the following:

The set $tuples_D(T_{\not\models})$ is:

$$\{(v_0, v_4, \text{“UE”}, v_5, \text{“Algebra I”}, v_6, \text{“SEM1”}, \text{“Dep”}),$$
$$(v_0, v_7, \text{“UE”}, v_8, \text{“Economics”}, v_9, \text{“SEM1”}, \text{“Main”})\}.$$

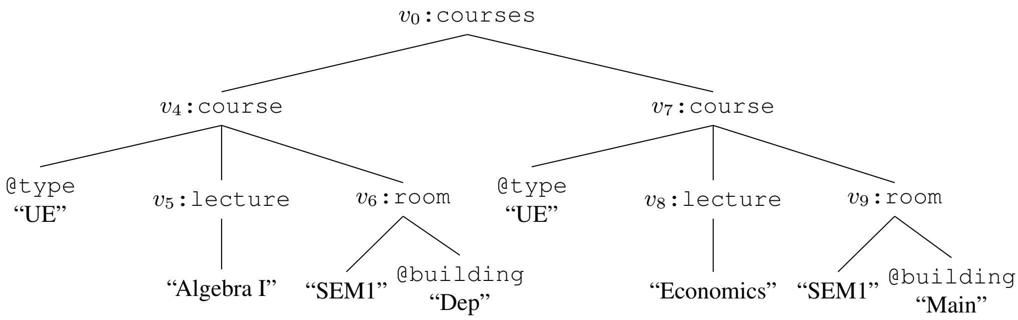


Figure 3.6: The XML tree $T_{\not\models}$ that violates the XFD given in Equation 3.1.

Let t_1 denote the first tuple from above and t_2 the second tuple. Now,

$$t_1.\texttt{courses.course.room.S} = t_2.\texttt{courses.course.room.S} = \text{"SEM1"},$$

but then

$$t_1.\texttt{courses.course.@building} = t_2.\texttt{courses.course.@building},$$

which is not the case, since $t_1.\texttt{courses.course.@building} = \text{"Dep"}$ and $t_2.\texttt{courses.course.@building} = \text{"Main"}$. ◁

As with FDs we define some additional notions. Let D be a DTD, let $\Sigma \subseteq \mathcal{F}D(D)$, and let $\varphi \in \mathcal{F}D(D)$. A DTD D together with a set of XFDs σ, denoted by (D, Σ), implies φ, denoted by $(D, \Sigma) \vdash \varphi$, if for any tree T, with $T \models D$ and $T \models \Sigma$, it is the case that $T \models \varphi$. We denote with $(D, \Sigma)^+$ the set of all XFDs implied by (D, Σ). We call an XFD φ *trivial* if $(D, \emptyset) \vdash \varphi$. For example, let $p \in paths(D)$ and $p.@l \in paths(D)$, then the XFD $p \to p.@l$ is a trivial.

3.3.2.1 Implication of XFDs

In this section we summarize the results on the complexity of the implication problem for XFDs. Remember that the implication problem for relational FDs can be decided in linear time (see Section 2.2.1.1). The implication problem of XFDs is much harder. In principle, checking if an XFD φ is not implied by a set of XFDs Σ, involves the construction of an XML tree, such that $T \models (D, \Sigma)$, but $T \not\models \varphi$. A proof for the existence of such a tree is a proof for the complement of the implication problem, which is the proof idea of the following theorem:

Theorem 3.1. *(Implication Problem for XFDs [3])*
The implication problem for XML functional dependencies over DTDs is solvable in co-NEXPTIME.

The high complexity is not due to the XFDs, but rather due to the complexity of DTDs. If we restrict the type of DTDs the implication problem can be solved more efficiently. We will give the results established by Arenas and Libkin in [3, 8] for two different types of DTDs.

Simple DTDs. The complexity of DTDs can be restricted through the complexity of the regular expressions in the production rules P. For this we define *trivial* regular expressions, which are, given an alphabet A, of the form $s_1, \ldots, s_n$, such that for each s_i there is a different letter $a_i \in A$ and s_i is either a_i or a_i? or a_i^+ or a_i^*. If the words of a regular expression are a permutation of a trivial regular expression, then this regular expression is called *simple*. All production rules in *simple* DTDs use simple regular expressions. Most real world DTDs are of this type [8].

Theorem 3.2. *(Implication Problem for XFDs over simple DTDs [8])*
The implication problem for XFDs over simple DTDs is solvable in quadratic time.

Relational DTDs. The second class of DTDs we introduce are relational DTDs. As we will see, the implication problem for this class is not tractable. A DTD D is called *relational* if for each XML tree T, such that $T \vDash D$, then for any nonempty subset X of $tuples_D(T)$, we can construct a set of trees $\mathcal{T}_X$ such that $\mathcal{T}_X \vDash D$. For example, the DTD `<!ELEMENT a (b,b)>` is not relational [8], because the tree built from just one of the tree tuples $\{(v_0, v_1), (v_0, v_2)\}$ does not satisfy the given DTD.

Theorem 3.3. *(Implication Problem for XFDs over relational DTDs [8])*
The implication problem for XFDs over relational DTDs is coNP-complete.

3.3.2.2 Nonaxiomatizability of XFDs

In Section 2.2.1.2 we gave an axiomatic system for relational FDs. Unfortunately, it is not possible to give an axiomatization for XFDs. First, we introduce some additional terms. Let D be a DTD and let Σ be a set of XFDs over D. (D, Σ) is *closed under implication* if for every φ over D such that $(D, \Sigma) \vdash \varphi$, then $\varphi \in \Sigma$. Moreover, (D, Σ) is *closed under k-ary implication* if for every φ over D, if there exists $\Sigma' \subseteq \Sigma$ such that $|\Sigma'| \leq k$ and $(D, \Sigma') \vdash \varphi$, it is the case that $\varphi \in \Sigma$ [8]. An axiomatization contains rules of the form *if* Γ *then* γ, where Γ and γ are FDs. Let k be an integer. If $|\Gamma| \leq k$, for any Γ that appears in the left-hand side of a rule, then we say that this set of rules is a k-ary axiomatization. The next proposition gives a necessary condition for the existence of a k-ary axiomatization.

A proof for the contrapositive of the next proposition, altered for XFDs, shows that the implication problem for XFDs does not admit a finite axiomatization.

Proposition 3.1. *(Proposition 7.5 of [8]) For every $k \geq 0$, if there is a k-ary ground axiomatization for the implication problem of XFDs, then for every DTD D and a set of XFDs over Σ over D, if (D, Σ) is closed under k-ary implication then (D, Σ) is closed under implication.*

This proposition was already proven in [1]. If we now want to show that there is no axiomatic system for XFDs, we use Proposition 3.1 and show that the necessary conditions for a k-axiomatic system are not fulfilled. This can be done by finding a DTD D and a set of functional dependencies Σ, which have the following properties:

- (D, Σ) is closed under k-ary implication, and
- (D, Σ) is not closed under implication.

Arenas and Libkin use this idea in order to establish a proof for the next theorem.

Theorem 3.4. *(Nonaxiomatizability of XFDs)*
The implication problem for XML functional dependencies is not finitely axiomatizable.

3.3.2.3 The Direct-Mapping of XFDs

We will now extend the direct mapping introduced in Section 3.2 with the translation of FDs into XFDs. Let FD be a set of FDs over the schema $G(A_1, \ldots, A_n)$, such that, without loss of generality, all FDs are of the form $X \to A$, where A is an attribute. Then the set Σ_{FD} of XFDs is defined as follows [8]:

- For each FD $A_{i_1}, \ldots, A_{i_m} \to A_i \in FD$,
$$\{db.G.@A_{i_1}, \ldots, db.G.@A_{i_m}\} \to db.G.@A_i \in \Sigma_{FD}.$$
- Additionally, to avoid duplicates,
$$\{db.G.@A_1, \ldots, db.G.@A_n\} \to db.G \in \Sigma_{FD}.$$

Definition 3.9. (Direct-Mapping of XFDs)
Let (G, FD) be a relational schema, then the direct-mapping to XML FDs is (D_G, Σ_{FD}) as previously defined. ◁

Example 3.11. We extend Example 3.6 to include the FDs given in Equations 2.1-2.3. The set $\Sigma_{\Sigma_{course}}$ of XFDs is defined as:

$$\texttt{db.courses.@room} \to \texttt{db.courses.@building} \tag{3.4}$$

$$\{\texttt{db.courses.@lecture}, \texttt{db.courses.@type}\} \to \texttt{db.courses.@room} \tag{3.5}$$

$$\texttt{db.courses.@room} \to \texttt{db.courses.@type} \tag{3.6}$$

$$\begin{aligned}&\{\texttt{db.courses.@lecture}, \texttt{db.courses.@type},\\ &\texttt{db.courses.@room}, \texttt{db.courses.@building}\} \to \texttt{db.courses}\end{aligned} \tag{3.7}$$

◁

3.4 Normal Forms

So far we have established the preliminaries to define a normal form for XML documents. In general, the goal is to generalize BCNF, which says that we should not represent the same fact twice. Therefore, we will first look at what a "redundancy" is in the context of XML documents and then, based on those insights, study XML Normal Form as defined in [8].

3.4.1 Redundancy in XML Documents

Let us consider the XML tree given in Figure 3.3. Each room has a name and is located in a specific building. Suppose a new course is added to this XML document. This course is also located in the seminar room "SEM1". Then, because of the XFD in Equation 3.1, we have to store the associated building "Dep" twice in this XML tree. Thus, the DTD together with the XFD in Equation 3.1 leads to redundancy. Such a redundancy is closely related to redundancies in relational data and can also be repaired in such a manner. We create new elements that store the rooms together with their building attribute. The lecture now just stores a reference to the room element. The resulting XML tree is illustrated in Figure 3.7. The next example illustrates a redundancy more closely related to the hierarchical structure of XML documents.

Example 3.12. (Example 1.2 from [8]) The DTD in Figure 3.8 is a part of the DBLP database [54] and stores data about conferences. In particular, this DTD stores the information of papers, which appeared in conference proceedings. Conference proceedings are in general distributed into several `issues`. Each issue contains several papers, stored in `inproceedings` elements. Those have a `@year` attribute. Now consider the following XFD, which says that any two `inproceedings` children of the same `issue` must have the same year:

$$\texttt{db.conf.issue} \rightarrow \texttt{db.conf.issue.inproceedings.@year}. \qquad (3.8)$$

This XFD leads to a redundancy. The `@year` attribute is stored several times per issue. This can be easily repaired by moving the `@year` attribute to the `issue` element. ◁

3.4.2 XML Normal Form

XML Normal Form (XNF) generalizes BCNF for XML documents [8]. Thus, it tries to avoid redundancies as described in the previous section. We need to express that we do not want to store implied data several times in an XML tree. This is captured by the following definition:

Definition 3.10. (XML Normal Form [8]) Given a DTD D and a set $\Sigma \subseteq \mathcal{F}D(D)$ of XFDs over D, (D, Σ) is in XML Normal Form (XNF) iff for every nontrivial XFD $\varphi \in (D, \Sigma)^+$, of the form $S \rightarrow p.@l$ or $S \rightarrow p.S$, it is the case that $S \rightarrow p$ is in $(D, \Sigma)^+$. ◁

Intuitively, this definition says that for all trees T conforming to a DTD D, whenever some set of paths S determines an attribute $@l$ of an element p, this attribute should only be stored once at exactly this element p. It is important that we only consider nontrivial XFDs, because the trivial FD $p.@l \rightarrow p.@l$ is always in $(D, \Sigma)^+$, but often $p.@l \rightarrow p \notin (D, \Sigma)^+$. We will now revisit the examples of the previous section.

Example 3.13. Let us first consider the DTD given in Figure 3.8 together with the XFD given in Equation 3.8. This DTD is not in XNF, because the XFD

$$\texttt{db.conf.issue} \rightarrow \texttt{db.conf.issue.inproceedings} \qquad (3.9)$$

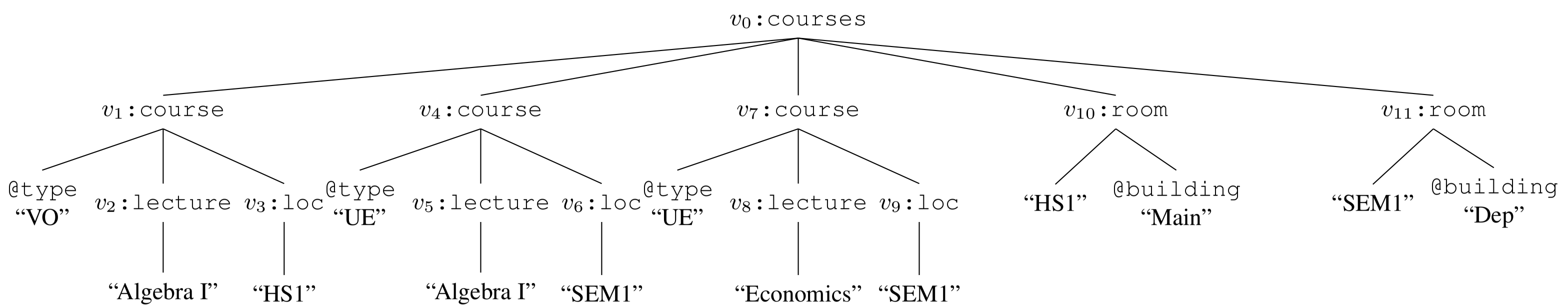


Figure 3.7: The tree representation of the XML document in Figure 3.3 with a new element room, that removes a redundancy.

```
<!DOCTYPE db [
  <!ELEMENT db (conf*)>
  <!ELEMENT conf (title, issue+)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT issue (inproceedings+)>
  <!ELEMENT inproceedings (author+, title)>
    <!ATTLIST inproceedings
          key ID #REQUIRED
          pages CDATA #REQUIRED
          year CDATA #REQUIRED>
  <!ELEMENT author (#PCDATA)>
]>
```

Figure 3.8: Part of the DTD from the DBLP database [54], taken from [8]

is not in $(D, \Sigma)^+$. The above XFD would imply that each `issue` element has only one `inproceedings` child element. The DTD, resulting from repair proposed in Example 3.12, is in XNF. During the repair, the XFD from Equation 3.8 is not altered but dropped, since `db.conf.issue` $\rightarrow$ `db.conf.issue.@year` is a trivial XFD. $\triangleleft$

Example 3.14. We consider the DTD in Figure 3.4, which is translated from the relational schema given in Figure 2.1. The translated XFDs $\Sigma_{\Sigma_{course}}$ are listed in Equations 3.4-3.7. The DTD together with its XFDs is not in XNF. There are two XNF violations:

- First, consider the XFD in Equation 3.4. The XFD `db.courses.@room` $\rightarrow$ `db.courses` is not in $(D_{course}, \Sigma_{\Sigma_{course}})^+$, which would imply that two different courses cannot be in the same room. If we store the rooms and their building separately, as illustrated in Figure 3.7, the XFD in Equation 3.4 no longer violates XNF.
- Additionally, also the XFD in Equation 3.6 leads to an XNF violation, because the XFD `db.courses.@room` $\rightarrow$ `db.courses` is not in $(D_{course}, \Sigma_{\Sigma_{course}})^+$. Notice that the original FD of the relational schema also leads to a BCNF violation.

As we have seen in Example 2.2, a repair of the corresponding relational schema changes the FDs, since $lecture, type \rightarrow room$ cannot be stated any longer. In XML we can use the hierarchical structure of XML documents to achieve a DTD that keeps all XFDs [4]. Such a DTD D_{XNF}, generated by the approach proposed in [49], is listed in Figure 3.9.

```
<!DOCTYPE db [
  <!ELEMENT db (t*,r*)>
  <!ELEMENT t (l*)>
  <!ATTLIST t
       type CDATA #REQUIRED>
  <!ELEMENT l (loc*)>
  <!ATTLIST l
       lecture CDATA #REQUIRED>
  <!ELEMENT loc EMPTY>
  <!ATTLIST loc
       room CDATA #REQUIRED>
  <!ELEMENT rooms EMPTY>
  <!ATTLIST rooms
       room CDATA #REQUIRED
       building CDATA #REQUIRED>
]>
```

Figure 3.9: A DTD originally mapped from the relational schema $course$, which is repaired to be in XNF

Let Σ_{XNF} be the following XFDs over D_{XNF}:

$$\texttt{db.t.@type} \rightarrow \texttt{db.t} \tag{3.10}$$

$$\{\texttt{db.t}, \texttt{db.t.l.@lecture}\} \rightarrow \texttt{db.t.l} \tag{3.11}$$

$$\{\texttt{db.t.l}, \texttt{db.t.l.loc.@room}\} \rightarrow \texttt{db.t.l.loc} \tag{3.12}$$

$$\{\texttt{db.t.l.@lecture}, \texttt{db.t.@type}\} \rightarrow \texttt{db.t.l.loc.@room} \tag{3.13}$$

$$\texttt{db.t.l.loc.@room} \rightarrow \texttt{db.t.@type} \tag{3.14}$$

$$\texttt{db.rooms.@room} \rightarrow \texttt{db.rooms} \tag{3.15}$$

$$\texttt{db.rooms.@room} \rightarrow \texttt{db.rooms.@building} \tag{3.16}$$

The idea of the DTD D_{XNF} together with the XFDs Σ_{XNF} is to first group together all courses of a particular type (see Equation 3.10). Then, we store the different lecture names with their locations (Equations 3.11 and 3.12). Equations 3.13 and 3.14 correspond to the original FDs $lecture, type \rightarrow room$ and $room \rightarrow type$, respectively. Equation 3.14 enforces that the same room cannot appear in different `t` subtrees. Finally, Equations 3.15 and 3.16 store that a room is located in a specific building.

It can be easily verified that the XFDs $\{\texttt{db.t.l.@lecture}, \texttt{db.t.@type}\} \rightarrow \texttt{db.t.l.loc}$, $\texttt{db.t.l.loc.@room} \rightarrow \texttt{db.t}$ and $\texttt{db.rooms.@room} \rightarrow \texttt{db.rooms}$ are in $(D_{XNF}, \Sigma_{XNF})^+$. Therefore (D_{XNF}, Σ_{XNF}) is in XNF. The redundancy-free XML tree with the same information as in Figure 2.1 is illustrated in Figure 3.10. $\triangleleft$

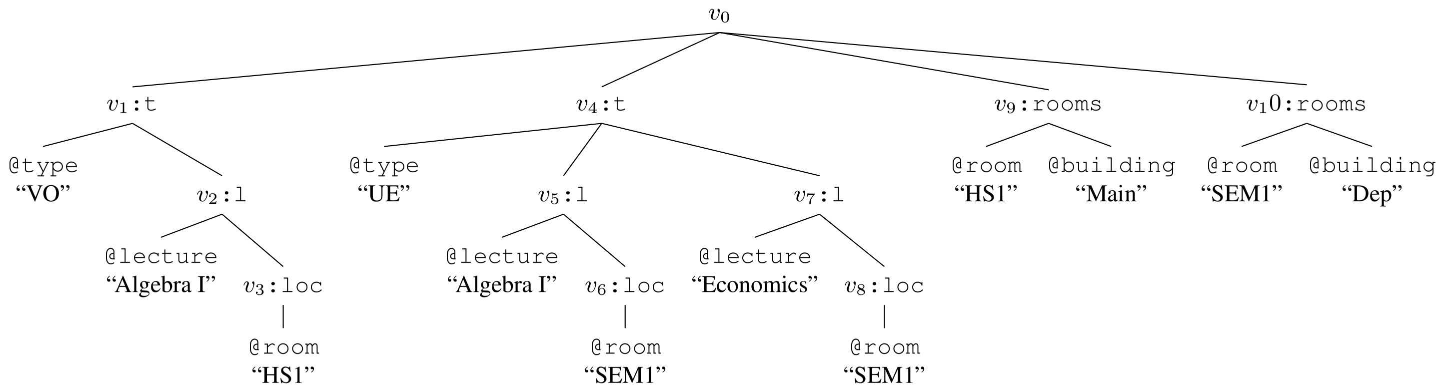


Figure 3.10: A redundancy free XML tree with the same information as in Figure 2.1.

The next theorem shows that BCNF and XNF are equivalent.

Theorem 3.5. *(BCNF and XNF, Proposition 5.5 of [8]) Let $G(A_1, \ldots, A_n)$ be a relational schema, FD be a set of functional dependencies over G. D_G is the DTD and Σ_{FD} is the set of XFDs, which are translated by the direct-mapping of relational schemas to XML. Then, (G, FD) is in BCNF iff (D_G, Σ_{FD}) is in XNF.*

Proof. The proof can be found in [8]. It follows from the fact that we can encode nested relation schemas into XML trees and that a normal form for nested relation schemas (NNF-FD) [57] coincides with XNF [8]. □

The Complexity of Testing XNF. By definition, testing XNF involves checking a condition on all XFDs implied by a DTD D and a set of XFDs Σ. Since Σ contains a finite number of paths, we can restrict every recursive DTD to a finite number of "unfoldings" of recursive rules [4]. Therefore it is possible to assume, without loss of generality, that DTDs are non-recursive. Then, we know that testing XNF is decidable. This follows from Theorem 3.1. If we restrict the class of DTDs we can first prove the following property.

Proposition 3.2. *[8] Given a relational DTD D and a set Σ of XFDs over D, then (D, Σ) is in XNF iff for each nontrivial XFD of the form $S \rightarrow p.@l$ or $S \rightarrow p.S$ in Σ, $S \rightarrow p \in (D, \Sigma)^+$.*

This proposition restricts the number of XFDs we have to check. Together with the complexity of the implication problem of simple and relational DTDs (see Theorems 3.2 and 3.3 respectively) the following follows:

Corollary 3.1. *[8] Testing if (D, Σ) is in XNF can be done in cubic time for simple DTDs, and is coNP-complete for relational DTDs.*

3.5 Summary

In this chapter we have introduced a normal form for XML documents. First, we presented a formal model for XML documents. We used DTDs to define the structure of an XML document. We showed that we can directly map relational schemas to DTDs and relational instances to XML documents. The information containted in an XML document can be represented using tree tuples. XML functional dependencies formulated over DTDs are evaluated over tree tuples. We can translate FDs over a relational schema to XFDs over a DTD, that is generated through the direct mapping from the relational schema. Such XFDs capture the semantic information of the original FDs. The implication problem for arbitrary DTDs is in co-NEXPTIME, in quadratic time for simple DTDs and coNP-complete for relational DTDs. Furthermore, we have shown that XFDs are nonaxiomatizable.

XFDs lead to redundancies in XML documents. A DTD in XML normal form tries to avoid redundancies. A DTD is in XNF if for every element e in an XML document that is implied by some other elements X it is the case that also the parent element of e is implied by the elements X. This generalizes the idea that these elements X must be a superkey. We have shown that a schema together with a set of FDs is in BCNF if and only if the DTD and a set of XFDs, both generated by the direct-mapping from the relational schema and FDs, is in XNF. This result shows that XNF is a generalization of BCNF. For relational it is possible to show that XNF testing is coNP-complete and for simple DTDs it can be done in cubic time.

CHAPTER 4

A New Normal Form For Description Logics

So far we have introduced and studied normal forms for relational data and XML documents. Both are technologies to store and exchange information. XML documents are mainly used in the World Wide Web. As the World Wide Web evolves to the Semantic Web [20, 64], information evolves as well. Information stored with the relational model is represented as tables, information stored in XML documents as trees, and the information stored in the Semantic Web as a graph. Therefore, the data models used in the Semantic Web can be seen as graph databases.

The W3C consortium started an initiative to standardize data models in the Semantic Web. As a result the Resource Description Framework (RDF) [44] was created as a formal language for describing structured information [45]. An RDF document can be viewed as a directed, labeled graph. Each resource is identified by a node. Nodes can be linked by directed edges. These edges are labeled, and describe the relationship between two nodes. The edge label is in RDF called *predicate*. Such a binary relationship between two resources is denoted in RDF by a *triple*. A triple consists of a *subject* s connected by a *predicate* p to an *object* o, which is denoted by (s, p, o). For example, the resource $p1$ in Figure 1.1c is connected via the $has\_article$ relation to the resource $a1$, which is denoted by the RDF triple:

$$(p1, has\_article, a1)$$

Additionally, semantic information can be attached to the nodes by means of an ontology. The most simple form of semantic information is to "type" resources. That is, we associate to a resource a specific class. This form of reasoning is already available in the RDF standard via the relation `rdf:type`, e.g. $(p1, \texttt{rdf:type}, \mathit{proc})$. In all figures we use UML like annotations of class membership, i.e. instead of a binary relation to a node denoting the class, we write $resource : \mathit{type}$. For example, in Figure 1.1c we write $p1 : \mathit{proc}$ to denote that the resource $p1$ is of type proc.

A simple ontology language is RDF Schema (RDF(S)) [22]. With RDF(S) we can attach even more semantic information to resources and roles. For example, we can specify that a resource is a class, e.g. $(\mathit{proc}, \texttt{rdf:type}, \texttt{rdfs:class})$. Furthermore, we can type predicates, i.e. we can specify domain and range of predicates. For example, the predicate $\mathit{has\_article}$ has the domain proc and the range $\mathit{article}$:

$$(\mathit{has\_article}, \texttt{rdfs:domain}, \mathit{proc})$$
$$(\mathit{has\_article}, \texttt{rdfs:range}, \mathit{article})$$

Such information enables us to infer new information. For example, the above together with the triple $(p1, \mathit{has\_article}, a1)$ infers:

$$(p1, \texttt{rdf:type}, \mathit{proc})$$
$$(a1, \texttt{rdf:type}, \mathit{article})$$

Even more semantic information can be attached to the data with the Web Ontology Language OWL [65] and OWL2 [59]. For example, OWL allows to state class disjointness. The semantics of these languages is best captured by Description Logics (DLs). Description Logics are a decidable fragment of first-order predicate logic and well-suited as a dependency language for graph databases. In this chapter we will discuss and introduce a normal form for DLs. We want to view redundancies and normal forms from a relational perspective. Therefore, we choose the DL *DL-Lite*$_{\mathcal{A}}$. *DL-Lite*$_{\mathcal{A}}$ is the formal model for the OWL2 profile OWL2 QL [58], which is particular well-suited to query data stored in relational databases. Additionally, *DL-Lite*$_{\mathcal{A}}$ has just enough expressivity for conceptual modeling [19, 23].

We will introduce in Section 4.1 the Description Logic *DL-Lite*$_{\mathcal{A}}$ as a formal model for graph-databases. Section 4.2 will then extend the "Direct Mapping of Relational Data to RDF" to map relational data into *DL-Lite*$_{\mathcal{A}}$ KBs. Data dependencies for *DL-Lite*$_{\mathcal{A}}$ are discussed in Section 4.3. Finally, a normal form for *DL-Lite*$_{\mathcal{A}}$ extended with data dependencies is proposed in Section 4.4 and Section 4.5 shows that this normal form is a generalization of BCNF.

4.1 Preliminaries

4.1.1 The Description Logic *DL-Lite*$_{\mathcal{A}}$

Description Logics (DLs) [13] were developed as a formal language for structured knowledge representation (KR). The goal of KR is to "develop formalisms for providing high-level descriptions of the world that can be effectively used to build intelligent applications" [12, 13]. DLs try to fulfill this goal. First, DLs provide us with a method to model important notions of a domain in terms of concept *descriptions*. The basic components of DLs are *concepts*, representing sets of objects and *roles*, which establish relationships between (instances of) concepts. The knowledge in DLs is separated into terminological knowledge, stored in an TBox and assertional knowledge, represented by an ABox. The TBox specifies general properties of concepts and roles. The ABox describes individual objects and their relationship. Second, the *logic*-based semantics of DLs allows us to infer new knowledge. These reasoning services include concept and role subsumption, knowledge base satisfiability and instance checking. The study of DLs in terms of expressivity and computational complexity of reasoning is one of the most important issues in DL research.

We will introduce *DL-Lite*$_{\mathcal{A}}$ [24, 60], a DL of the *DL-Lite* family [10, 26, 28]. The advantage of the *DL-Lite* family is its low complexity of reasoning. For example, instance checking and query answering can be done in LOGSPACE with respect to data complexity. Still it is possible to capture conceptual data models and object-oriented formalisms [24].

4.1.1.1 Syntax of *DL-Lite*$_{\mathcal{A}}$

In comparison to the other DLs in the *DL-Lite* family, *DL-Lite*$_{\mathcal{A}}$ distinguishes between *objects* and *values*. Therefore, our domain of interest is represented in terms of *concepts* and *roles*. Additionally, we introduce *value-domains* which denote a set of (data) values and *attributes*, which denote a binary relation between objects and values. The building blocks of *DL-Lite*$_{\mathcal{A}}$ are *atomic* concepts A,A_1,. . .,A_n, atomic roles P,P_1,. . .,P_n and atomic attributes U,U_1,. . .,U_n. All of them are denoted by a name. All names of attributes, and no other names, start with an @. From these we build complex concepts, roles, value-domains and attributes according to the following syntax:

	atomic	basic	arbitrary
concept	A	$B \to A \mid \exists Q \mid \delta(U)$	$C \to B \mid \neg B$
role	P	$Q \to P \mid P^-$	$R \to Q \mid \neg Q$
value-domain		$E \to \rho(U)$	$F \to \top_D \mid T_1 \mid \ldots \mid T_n$
attribute	U	$V \to U$	$W \to V \mid \neg V$

We denote by B,B_1,. . .,B_n *basic* concepts. A basic concept is either an atomic concept, the *domain* of a role Q ($\exists Q$), also called unqualified existential restriction, or the domain of an attribute U ($\delta(U)$). An *arbitrary* concept, denoted by C,C_1,. . .,C_n, is built from a basic concept or its *negation*.

A *basic* role, denoted by Q,Q_1,. . .,Q_n, is either an atomic role or the *inverse* of an atomic role (P^-). An *arbitrary* role can in addition to a basic role also be the negation of a basic role. In the following, when Q is a basic role, the expression Q^- stands for P^- when $Q = P$, and for P when $Q = P^-$.

A *basic* value-domain E is given by the *range* of an atomic attribute U. *Arbitrary* value-domains are either the universal value-domain $\top_D$, or one of n pairwise disjoint unbounded value-domains $T_1, \ldots, T_n$, which correspond to RDF data types, such as `xsd:string`, etc.

Example 4.1. Let us model the same information given in Figure 2.1. We need the following atomic concepts: $course$ represents a course, $type$ a course type, $room$ a room, and $building$ a building. We connect these atomic concepts using the atomic roles: $located$, has_room and for. With the atomic attribute $@name$ we attach a name value to different concepts. ◁

With those expressions it is possible, like in any other DL, to represent the domain of discourse in terms of a knowledge base $\mathcal{K}$. The KB $\mathcal{K}$ has two components. The TBox $\mathcal{T}$ consists of a finite set of intensional assertions. Intensional assertions describe the world in more general terms, for example "Birds can fly.". The ABox $\mathcal{A}$ consists of a finite set of extensional assertions, which describe individuals, for example "Tweety is a bird". Therefore, we often write $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. The TBox $\mathcal{T}$ consists of assertions of the form:

$B \sqsubseteq C$	concept inclusion
$Q \sqsubseteq R$	role inclusion
$E \sqsubseteq F$	value-domain inclusion
$U \sqsubseteq V$	attribute inclusion
$(\text{funct } Q)$	role functionality
$(\text{funct } U)$	attribute functionality

Intuitively, the above inclusion assertions state that each instance of the concept, role, value-domain or attribute on the left-hand side is also an instance of the right-hand side. We call inclusion assertions without negation ("¬") on the right-hand side *positive inclusions* (PIs), and the others *negative inclusions* (NIs). For example, the RDF triple `<has_article> rdfs:domain <proc>` would be represented by the assertion $\mathit{proc} \sqsubseteq \exists \mathit{has\_article}$.

Functionality assertions ($(\text{funct } Q)$ or $(\text{funct } U)$) express that in every model of $\mathcal{T}$ the first component of a role (or attribute) determines the second component, i.e. this binary relation is a function.

The following conditions must be satisfied by every *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}$. These are crucial for the tractability of reasoning in *DL-Lite*$_{\mathcal{A}}$ [60]:

- for each atomic role P, if either $(\text{funct } P)$ or $(\text{funct } P^-)$ occurs in $\mathcal{T}$, then $\mathcal{T}$ does not contain assertions of the form $Q' \sqsubseteq P$ or $Q' \sqsubseteq P^-$, where Q' is a basic role.
- for each atomic attribute U, if $(\text{funct } U)$ occurs in $\mathcal{T}$, then $\mathcal{T}$ does not contain assertions of the form $U' \sqsubseteq U$, where U' is an atomic attribute.

Example 4.2. Let us model a TBox $\mathcal{T}_c$ for the information given in Figure 2.1. We will use the concepts and roles given in Example 4.1.

$$
\begin{array}{ccc}
\mathit{room} \sqsubseteq \exists \mathit{for} & \exists \mathit{for} \sqsubseteq \mathit{room} & \exists \mathit{for} \sqsubseteq \mathit{type} \\
\mathit{course} \sqsubseteq \exists \mathit{located} & \exists \mathit{located} \sqsubseteq \mathit{course} & \exists \mathit{located}^- \sqsubseteq \mathit{room} \\
\mathit{room} \sqsubseteq \exists \mathit{has\_room}^- & \exists \mathit{has\_room} \sqsubseteq \mathit{building} & \exists \mathit{has\_room}^- \sqsubseteq \mathit{room} \\
\mathit{course} \sqsubseteq \neg \mathit{room} & \mathit{course} \sqsubseteq \neg \mathit{type} & \mathit{course} \sqsubseteq \neg \mathit{building} \\
\mathit{room} \sqsubseteq \neg \mathit{building} & \mathit{room} \sqsubseteq \neg \mathit{type} & \mathit{building} \sqsubseteq \neg \mathit{type} \\
\mathit{course} \sqsubseteq \delta(@\mathit{name}) & \mathit{type} \sqsubseteq \delta(@\mathit{name}) & \rho(@\mathit{name}) \sqsubseteq \mathtt{xsd:string} \\
\mathit{room} \sqsubseteq \delta(@\mathit{name}) & \mathit{building} \sqsubseteq \delta(@\mathit{name}) &
\end{array}
$$

$$(\text{funct } \mathit{for}) \qquad\qquad (\text{funct } \mathit{located})$$
$$\left(\text{funct } \mathit{has\_room}^-\right) \qquad\qquad (\text{funct } @\mathit{name}) \qquad \triangleleft$$

So far we have defined expressions and assertions that represent the domain of discourse in general. The *DL-Lite*$_{\mathcal{A}}$ ABox allows us to express properties about different individuals. First, we need to define constants that represent such individuals. Let this set of constants be denoted by Γ. Γ is partitioned into two sets, Γ_V (for the set of constant symbols for values) and Γ_O (for the set of constant symbols for objects). A *DL-Lite*$_{\mathcal{A}}$ ABox consists of a finite set of *membership assertions* of the form

$$A(a), \qquad P(a,b), \qquad U(a,v),$$

where A, P, and U are an atomic concept, atomic role and atomic attribute, respectively. The constant symbols a and b are from Γ_O and v is from Γ_V.

We will denote by $\mathit{sign}^C(\mathcal{K})$ the set of all atomic concepts in a KB $\mathcal{K}$ and by $\mathit{sign}^R(\mathcal{K})$ the set of all atomic roles. We call $\mathit{sign}(\mathcal{K}) = \mathit{sign}^C(\mathcal{K}) \cup \mathit{sign}^R(\mathcal{K})$ the signature of a KB $\mathcal{K}$.

Example 4.3. We continue Example 4.2 and specify membership assertions to represent the information given in Figure 2.1. We have the following sets of constant symbols:

- $\{c1, c2, c3, t1, t2, r1, r2, b1, b2\} \subseteq \Gamma_O$
- {“Algebra I”, “Economics I”, “VO”, “UE”, “SEM1”, “HS1”, “Main”, “Dep”} $\subseteq \Gamma_V$

The *DL-Lite*$_{\mathcal{A}}$ ABox $\mathcal{A}_c$ consists of the following assertions:

$\mathit{course}(c1)$	$\mathit{type}(t1)$	$\mathit{room}(r1)$
$\mathit{course}(c2)$	$\mathit{type}(t2)$	$\mathit{room}(r2)$
$\mathit{course}(c3)$	$\mathit{building}(b1)$	$\mathit{building}(b2)$
$@\mathit{name}(c1,$ “Algebra I”$)$	$@\mathit{name}(t1,$ “VO”$)$	$@\mathit{name}(r1,$ “HS1”$)$
$@\mathit{name}(c2,$ “Algebra I”$)$	$@\mathit{name}(t2,$ “UE”$)$	$@\mathit{name}(r2,$ “SEM1”$)$
$@\mathit{name}(c3,$ “Economics I”$)$	$@\mathit{name}(b1,$ “Main”$)$	$@\mathit{name}(b2,$ “Dep”$)$
$\mathit{located}(c1, r1)$	$\mathit{located}(c2, r2)$	$\mathit{located}(c3, r2)$
$\mathit{for}(r1, t1)$	$\mathit{for}(r2, t2)$	
$\mathit{has\_room}(b1, r1)$	$\mathit{has\_room}(b2, r2)$	

The *DL-Lite*$_{\mathcal{A}}$ ABox $\mathcal{A}_c$ can also be represented as a graph. Such a graph is depicted in Figure 4.1. The KB $\mathcal{K}_c = \langle \mathcal{T}_c, \mathcal{A}_c \rangle$ is a representation of the same information as given in Figure 2.1. $\triangleleft$

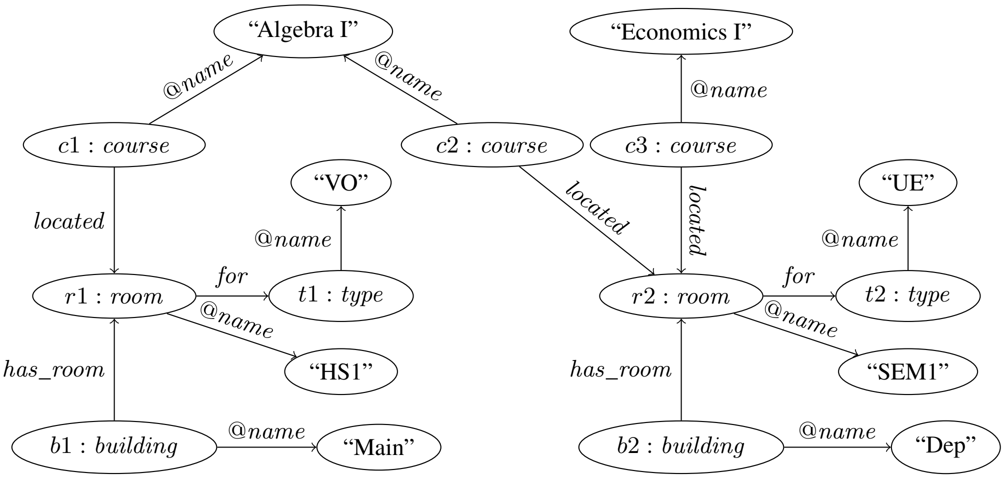


Figure 4.1: A graph representation of the ABox $\mathcal{A}_c$.

4.1.1.2 Semantics of *DL-Lite*$_\mathcal{A}$

Now that we have fully defined the syntax of *DL-Lite*$_\mathcal{A}$ we need to add meaning to the expressions and assertions. We define the semantics of *DL-Lite*$_\mathcal{A}$ in terms of interpretations, which are first order structures. A *DL-Lite*$_\mathcal{A}$ *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consists of an *interpretation domain* $\Delta^\mathcal{I}$ and an *interpretation function* $\cdot^\mathcal{I}$. The interpretation domain $\Delta^\mathcal{I}$ is the disjoint union of two non-empty sets: the *domain of objects* $\Delta^\mathcal{I}_O$ and the *domain of values* $\Delta^\mathcal{I}_V$. The interpretation function $\cdot^\mathcal{I}$ assigns an element of $\Delta^\mathcal{I}$ to each constant in Γ, such that for all $a \in \Gamma_O$, $a^\mathcal{I} \in \Delta^\mathcal{I}_O$ and for all $c \in \Gamma_V$, $c^\mathcal{I} \in \Delta^\mathcal{I}_V$. The DL *DL-Lite*$_\mathcal{A}$ adopts the *unique name assumption* (UNA), therefore we also assume that for each pair $a_1, a_2 \in \Gamma$, whenever $a_1 \neq a_2$, we have that $a_1^\mathcal{I} \neq a_2^\mathcal{I}$. Additionally, the interpretation function $\cdot^\mathcal{I}$ maps

- atomic concepts to subsets of the interpretation domain of objects, i.e.

$$A^\mathcal{I} \subseteq \Delta^\mathcal{I}_O,$$

- atomic roles to subsets of the crossproduct of the interpretation domain of objects, i.e.

$$P^\mathcal{I} \subseteq \Delta^\mathcal{I}_O \times \Delta^\mathcal{I}_O,$$

- atomic attributes to subsets of the crossproduct of the interpretation domain of objects and values, i.e.

$$U^\mathcal{I} \subseteq \Delta^\mathcal{I}_O \times \Delta^\mathcal{I}_V,$$

- value-domains to subsets of the interpretation domain of values, i.e.

$$T_i^\mathcal{I} \subseteq \Delta^\mathcal{I}_V \qquad\qquad \top_D^\mathcal{I} = \Delta^\mathcal{I}_V.$$

All interpretations of a particular KB agree on the semantics of each value-domain T_i and each constant in Γ_V. That is, each value domain T_i is interpreted as the set of values $val\,(T_i)$ corresponding to the RDF data type, and each constant $c_i \in \Gamma_V$ is interpreted as one specific value, denoted by $val\,(c_i)$, in $val\,(T_i)$.

For complex concepts, complex roles and complex attributes the interpretation has to satisfy the following conditions (given that $o, o' \in \Delta_O^{\mathcal{I}}$ and $v \in \Delta_V^{\mathcal{I}}$):

$$
\begin{aligned}
(\exists Q)^{\mathcal{I}} &= \{o \mid \exists o'.\,(o,o') \in Q^{\mathcal{I}}\} & \left(P^{-}\right)^{\mathcal{I}} &= \{(o,o') \mid (o',o) \in P^{\mathcal{I}}\} \\
(\delta\,(U))^{\mathcal{I}} &= \{o \mid \exists v.\,(o,v) \in U^{\mathcal{I}}\} & (\neg Q)^{\mathcal{I}} &= \left(\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}\right) \setminus Q^{\mathcal{I}} \\
(\neg B)^{\mathcal{I}} &= \Delta_O^{\mathcal{I}} \setminus B^{\mathcal{I}} & (\neg V)^{\mathcal{I}} &= \left(\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}\right) \setminus V^{\mathcal{I}} \\
(\rho\,(U))^{\mathcal{I}} &= \{v \mid \exists o.\,(o,v) \in U^{\mathcal{I}}\}
\end{aligned}
$$

We now turn our attention to the assertions in a KB TBox and ABox. Let α be a TBox assertion. Then, we say an interpretation $\mathcal{I}$ *satisfies* the TBox assertion α, denoted by $\mathcal{I} \vDash \alpha$, as follows:

- let $\alpha = \alpha_1 \sqsubseteq \alpha_2$, then $\mathcal{I} \vDash \alpha_1 \sqsubseteq \alpha_2$, if $\alpha_1^{\mathcal{I}} \subseteq \alpha_2^{\mathcal{I}}$,
- let $\alpha =$ (funct β), where β is either P, P^{-}, U. Then, $\mathcal{I} \vDash$ (funct β), if $(o, e_1) \in \beta^{\mathcal{I}}$ and $(o, e_2) \in \beta^{\mathcal{I}}$ implies $e_1 = e_2$, for each $o \in \Delta_O^{\mathcal{I}}$, and e_1, e_2 in either $\Delta_O^{\mathcal{I}}$ or $\Delta_V^{\mathcal{I}}$.

Let α be an ABox assertion. Then, we say an interpretation $\mathcal{I}$ *satisfies* the ABox assertion α, denoted by $\mathcal{I} \vDash \alpha$, as follows:

- let $\alpha = A(a)$, then $\mathcal{I} \vDash A(a)$, if $a^{\mathcal{I}} \in A^{\mathcal{I}}$,
- let $\alpha = P(a, a')$, then $\mathcal{I} \vDash P(a, a')$, if $(a^{\mathcal{I}}, a'^{\mathcal{I}}) \in P^{\mathcal{I}}$,
- let $\alpha = U(a, c)$, then $\mathcal{I} \vDash U(a, c)$, if $(a^{\mathcal{I}}, c^{\mathcal{I}}) \in U^{\mathcal{I}}$.

We say an interpretation $\mathcal{I}$ is a *model* of a *DL-Lite*$_{\mathcal{A}}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, denoted by $\mathcal{I} \vDash \mathcal{K}$, if $\mathcal{I}$ satisfies all the assertions in $\mathcal{T}$ and $\mathcal{A}$.

Example 4.4. We will give an interpretation $\mathcal{I}_c$ for the KB given in Example 4.3. The interpretation $\mathcal{I}$ is constituted from a domain $\Delta^{\mathcal{I}_c}$ and an interpretation function $\cdot^{\mathcal{I}_c}$. The domain consists of:

- Domain of objects $\Delta_O^{\mathcal{I}_c} = \{c1, c2, c3, t1, t2, r1, r2, b1, b2\}$, and
- Domain of values $\Delta_V^{\mathcal{I}_c} = \{$“Algebra I”, “Economics I”, “VO”, “UE”, “SEM1”, “HS1”, “Main”, “Dep”$\}$.

The interpretation function $\cdot^{\mathcal{I}_c}$ maps all constant symbols in Γ to the corresponding values in $\Delta^{\mathcal{I}_c}$, i.e. $e^{\mathcal{I}_c} = e$ for all $e \in \Gamma$. It is easily verified that this interpretation satisfies the KB $\mathcal{K}_c$, i.e. $\mathcal{I}_c \vDash \mathcal{K}_c$. ◁

We will now introduce the notion of an ABox seen as an interpretation, denoted by $DB(\mathcal{A})$.

Definition 4.1. (ABox interpretation $DB(\mathcal{A})$ [23]) Let $\mathcal{A}$ be a *DL-Lite*$_{\mathcal{A}}$ ABox. We denote by $DB(A) = \langle \Delta^{DB}(\mathcal{A}), \cdot^{DB}(\mathcal{A}) \rangle$ the interpretation defined as follows:

- $\Delta^{DB}(\mathcal{A})$ is the non-empty set consisting of the union of the set of all object constant occurring in $\mathcal{A}$ and the set $val(c)$, such that c is a value constant that occurs in $\mathcal{A}$,
- $a^{DB}(\mathcal{A}) = a$, for each object constant a,
- $A^{DB}(\mathcal{A}) = \{a \mid A(a) \in \mathcal{A}\}$, for each atomic concept A,
- $P^{DB}(\mathcal{A}) = \{(a_1, a_2) \mid P(a_1, a_2) \in \mathcal{A}\}$, for each atomic role P,
- $U^{DB}(\mathcal{A}) = \{(a, val(c)) \mid U(a, c) \in \mathcal{A}\}$, for each atomic attribute U. $\lhd$

It is clear that such an interpretation satisfies all ABox assertions, i.e. $DB(\mathcal{A}) \models \mathcal{A}$. Notice that the interpretation $\mathcal{I}_c$ given in Example 4.4, is exactly the ABox interpretation $DB(\mathcal{A}_c)$, where $\mathcal{A}_c$ is the ABox given in Example 4.3.

4.1.1.3 Queries over *DL-Lite*$_{\mathcal{A}}$ KB

We introduce here queries of *DL-Lite*$_{\mathcal{A}}$ KBs. A first-order query q over a *DL-Lite*$_{\mathcal{A}}$ KB $\mathcal{K}$ is a, possibly open, first-order logic formula (FOL) $\varphi(\mathbf{x})$. Such a query is built from atoms, which are in the case of queries over a *DL-Lite*$_{\mathcal{A}}$ KB the following:

- atomic concepts, written as $A(x)$,
- value-domains, written as $D(x)$,
- atomic roles, written as $P(x, y)$,
- atomic attributes, written as $U(x, y)$, or
- equality of variables, i.e. $x = y$.

The variables x, y are either variables in $\mathbf{x}$ or constants in Γ. The free variables $\mathbf{x}$ of $\varphi(\mathbf{x})$ form a tuple of (pairwise distinct) variables. The arity of a query is given by the arity of $\mathbf{x}$. A *boolean query* is a query with arity 0. We will only consider *conjunctive* (CQ) and *union of conjunctive* (UCQ) queries, which are of the form:

$$\begin{aligned} q(\mathbf{x}) &\leftarrow \exists \mathbf{y}_1 . conj_1(\mathbf{x}, \mathbf{y}_1) \\ &\vdots \\ q(\mathbf{x}) &\leftarrow \exists \mathbf{y}_n . conj_n(\mathbf{x}, \mathbf{y}_n) \end{aligned}$$

where $conj_k(\mathbf{x}, \mathbf{y}_k)$ is a conjunction of atoms. The free variables $\mathbf{x}$ are also called *distinguished* variables and the existentially quantified variables $y_1, \ldots, y_n$ are called *non-distinguished* variables. A CQ is a UCQ with no disjunction.

Let $\mathcal{I}$ be an interpretation over a *DL-Lite*$_\mathcal{A}$ KB. The answer to a UCQ $q = \varphi(\mathbf{x})$ is the set $q^\mathcal{I}$ of tuples $\mathbf{o} = \Delta^\mathcal{I} \times \cdots \times \Delta^\mathcal{I}$ such that the formula φ evaluates to true in $\mathcal{I}$ under the assignment that assigns each object in $\mathbf{o}$ to the corresponding variable in $\mathbf{x}$ [23]. The set of tuples $q^\mathcal{I}$ is called the answers to q over $\mathcal{I}$.

Example 4.5. Consider the KB from Example 4.3 and let $\mathcal{I}_c$ be the interpretation given in Example 4.4. The following query asks for the room and the type of a course:

$$q_1(r, l) \leftarrow \exists c.\mathit{is\_of\_type}(c, t) \wedge \mathit{located}(c, l).$$

The answers to q_1 are:

$$q_1^{\mathcal{I}_c} = \{(r1, t1), (r2, t2)\}. \qquad \triangleleft$$

The above notion of CQs only considers queries over a particular model of a KB. If we query a KB, we rather look for an answer over all possible interpretations. Such answers are called *certain* and we define them as follows.

Definition 4.2. (Certain answers [23]) Let $\mathcal{K}$ be a *DL-Lite*$_\mathcal{A}$ KB and q a UCQ over $\mathcal{K}$. A tuple $\mathbf{c}$ of constants appearing in $\mathcal{K}$ is a certain answer to q over $\mathcal{K}$, written $\mathbf{c} \in \mathit{cert}(q, \mathcal{K})$, if for every model $\mathcal{I}$ of $\mathcal{K}$, we have that $\mathbf{c}^\mathcal{I} \in q^\mathcal{I}$. $\triangleleft$

4.1.1.4 Reasoning in *DL-Lite*$_\mathcal{A}$

DLs provide different reasoning services. Among them, the most important, also for *DL-Lite*$_\mathcal{A}$ KB, are [23]:

- *KB satisfiability*: Given a KB $\mathcal{K}$, verify whether $\mathcal{K}$ admits at least one model
- *Concept and role satisfiability*: Given a TBox $\mathcal{T}$ and a concept C (resp. a role R), verify whether $\mathcal{T}$ admits a model $\mathcal{I}$ such that $C^\mathcal{I} \neq \emptyset$ (resp. $R^\mathcal{I} \neq \emptyset$).
- *Logical implication of an assertion*: A KB $\mathcal{K}$ logically implies an assertion α, denote $\mathcal{K} \models \alpha$, if every model of $\mathcal{K}$ satisfies α. Different types of assertions give different subproblems: *instance checking* ($\mathcal{K} \models C(a)$ or $\mathcal{K} \models R(a_1, a_2)$), *subsumption of concepts or roles* ($\mathcal{K} \models C_1 \sqsubseteq C_2$ or $\mathcal{K} \models R_1 \sqsubseteq R_2$) or *checking functionality* ($\mathcal{K} \models (\text{funct } Q)$).

Notice that in *DL-Lite*$_\mathcal{A}$ concept and role satisfiability, and logical implication of an assertion can be reduced to KB satisfiability (see [23] for details). Additionally, we are interested in:

- *Query answering*: Given a KB $\mathcal{K}$ and a query q over $\mathcal{K}$, compute the set $\mathit{cert}(q, \mathcal{K})$.

4.1.2 Reasoning over *DL-Lite*$_{\mathcal{A}}$ KB

In DLs reasoning is an important task. We have presented several reasoning services, all of which, except of query answering, can be reduced to KB satisfiability. Therefore, we will provide an introduction to KB satisfiability in *DL-Lite*$_{\mathcal{A}}$ as it is introduced in [23]. However, for the ease of presentation, we will make some simplifying assumptions. Those assumptions do not affect the generality of the presented results. First, we will not distinguish between concepts and values. Thus, our KBs contain only object constants, concepts and roles. Second, we will discard inclusion assertions of the form $B \sqsubseteq \top$. They do not have an impact on the semantics [23].

We will define the notion of a *canonical interpretation* of a *DL-Lite*$_{\mathcal{A}}$ KB. Such an interpretation is constructed according to the notion of *restricted chase* [1, 47]. This is done by first defining the notion of applicable assertions, then we define the chase for a *DL-Lite*$_{\mathcal{A}}$ KB, and finally, we use the chase to define the canonical interpretation. In this thesis, we will make use of the following simplifying notation for a basic role Q and two constants a_1, a_2:

$$Q\left(a_1, a_2\right) \text{denotes} \begin{cases} P\left(a_1, a_2\right), & \text{if } Q = P, \\ P\left(a_2, a_1\right), & \text{if } Q = P^-. \end{cases}$$

Definition 4.3. (Applicable PIs [23]) Let $\mathcal{S}$ be a set of *DL-Lite*$_{\mathcal{A}}$ membership assertions. Then, a PI α is applicable in $\mathcal{S}$ to a membership assertion $\beta \in \mathcal{S}$ if

- $\alpha = A_1 \sqsubseteq A_2$, $\beta = A_1(a)$, and $A_2(a) \notin \mathcal{S}$;
- $\alpha = A \sqsubseteq \exists Q$, $\beta = A(a)$, and there does not exist any constant a' such that $Q\left(a, a'\right) \in \mathcal{S}$;
- $\alpha = \exists Q \sqsubseteq A$, $\beta = Q(a, a')$ for some a', and $A(a) \notin \mathcal{S}$;
- $\alpha = \exists Q_1 \sqsubseteq \exists Q_2$, $\beta = Q_1(a_1, a_2)$ for some a_2, and there does not exist any constant a_2' such that $Q_2(a_1, a_2') \in \mathcal{S}$;
- $\alpha = Q_1 \sqsubseteq Q_2$, $\beta = Q_1(a_1, a_2)$, and $Q_2(a_1, a_2) \notin \mathcal{S}$. $\triangleleft$

Applicable PIs can be used to extend a *DL-Lite*$_{\mathcal{A}}$ model in order to satisfy all positive inclusion assertions of a *DL-Lite*$_{\mathcal{A}}$ TBox. The chase consists of a possibly infinite number of chase steps. It starts with a *DL-Lite*$_{\mathcal{A}}$ ABox $\mathcal{A}$. In each step a PI α is applied to a membership assertion β. Thus, we add a new membership assertion to $\mathcal{A}$, such that the PI α is not applicable to β anymore. It is clear that this process strongly depends on the order of the applied PIs. We will assume the order to be fixed. This can be done by assuming an infinite set Γ_N of lexicographically ordered constant symbols not occurring in $\mathcal{A}$ and a lexicographic ordering on the set of PIs. Now we can introduce the notion of the chase.

Definition 4.4. (The *DL-Lite*$_{\mathcal{A}}$ chase [23]) Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_{\mathcal{A}}$ KB, let $\mathcal{T}_p$ be the set of positive inclusion assertions in $\mathcal{T}$, let n be the number of membership assertions in $\mathcal{A}$, and let Γ_N be the set of lexicographically ordered constants not in $\mathcal{A}$. Assume that the membership assertions in $\mathcal{A}$ are numbered from 1 to n following their lexicographic order, and consider the following definition of sets $\mathcal{S}_j$ of membership assertions:

- $\mathcal{S}_0 = \mathcal{A}$
- $\mathcal{S}_{j+1} = \mathcal{S}_j \cup \{\beta_{new}\}$, where β_{new} is a membership assertion numbered with $n + j + 1$ in $\mathcal{S}_{j+1}$ and obtained as follows:
  - **let** β be the first membership assertion in $\mathcal{S}_j$ such that there exists a PI $\alpha \in \mathcal{T}_p$ applicable in $\mathcal{S}_j$ to β
  - **let** α be the lexicographically first PI applicable in $\mathcal{S}_j$ to β
  - **let** a_{new} be the constant of Γ_N that follows lexicographically all constants in $\mathcal{S}_j$
  - **case** α, β **of**

(cr1)	$\alpha = A_1 \sqsubseteq A_2$	and $\beta = A_1(a)$	**then** $\beta_{new} = A_2(a)$
(cr2)	$\alpha = A \sqsubseteq \exists Q$	and $\beta = A(a)$	**then** $\beta_{new} = Q(a, a_{new})$
(cr3)	$\alpha = \exists Q \sqsubseteq A$	and $\beta = Q(a, a')$	**then** $\beta_{new} = A(a)$
(cr4)	$\alpha = \exists Q_1 \sqsubseteq \exists Q_2$	and $\beta = Q_1(a, a')$	**then** $\beta_{new} = Q_2(a, a_{new})$
(cr5)	$\alpha = Q_1 \sqsubseteq Q_2$	and $\beta = Q_1(a, a')$	**then** $\beta_{new} = Q_2(a, a')$

Then, we call chase of $\mathcal{K}$, denoted $chase\,(\mathcal{K})$, the set of membership assertions obtained by the infinite union of all $\mathcal{S}_j$, i.e.,

$$chase\,(\mathcal{K}) = \bigcup_{j \in \mathbb{N}} \mathcal{S}_j. \qquad \triangleleft$$

It is now possible to define the canonical interpretation of a *DL-Lite*$_{\mathcal{A}}$ KB.

Definition 4.5. (Canonical interpretation ($can\,(\mathcal{K})$)) The *canonical interpretation* $can\,(\mathcal{K}) = \left\langle \Delta^{can(\mathcal{K})}, \cdot^{can(\mathcal{K})} \right\rangle$ is the interpretation where:

- $\Delta^{can(\mathcal{K})} = \Gamma_O \cup \Gamma_N$,
- $a^{can(\mathcal{K})} = a$, for each constant a occurring in $chase\,(\mathcal{K})$,
- $A^{can(\mathcal{K})} = \{a \mid A(a) \in chase\,(\mathcal{K})\}$, for each atomic concept A, and
- $Q^{can(\mathcal{K})} = \{(a_1, a_2) \mid P(a_1, a_2) \in chase\,(\mathcal{K})\}$, for each atomic role P. $\triangleleft$

The following property, proven in [23], holds for $can\,(\mathcal{K})$.

Lemma 4.1. *(Lemma 4.5 of [23]) Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{\mathcal{A}}$ KB and let $\mathcal{T}_p$ be the set of positive inclusion assertions in $\mathcal{T}$. Then, $can\,(\mathcal{K})$ is a model of $\langle \mathcal{T}_p, \mathcal{A} \rangle$.*

Since $can\,(\mathcal{K})$ is a model of $\langle \mathcal{T}_p, \mathcal{A} \rangle$, we can conclude that each KB, with only PIs in the TBox, is satisfiable. We now need to extend satisfiability to account for functional assertions and negative inclusion assertions. For functionality assertions satisfiability is easy to check. The following lemma says that we just need to verify if the interpretation $DB\,(\mathcal{A})$ satisfies the functionality assertions.

Lemma 4.2. *(Lemma 4.6 of [24]) Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ KB, and let $\mathcal{T}_f$ be the set of functionality assertions in $\mathcal{T}$. Then, $can\,(\mathcal{K})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$ if and only if $DB\,(\mathcal{A})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$.*

Proof sketch.

($\Rightarrow$) Follows from the fact that $\mathcal{A} \subseteq chase\,(\mathcal{K})$.

($\Leftarrow$) In each chase step we choose an applicable PI α. Only rules **cr2**, **cr4** and **cr5** can lead to a violation of a functionality assertion. Due to the restriction on *DL-Lite*$_\mathcal{A}$ KBs, rule **cr5** cannot lead to a violation of a functional dependency. Assume the new constant a_{new} introduced by rule **cr2** or **cr4** leads to a violation of a functionality assertion. This is only the case if there is a constant symbol a' such that $Q(a, a') \in \mathcal{S}$ or $Q_2(a, a') \in \mathcal{S}$. But then α would not have been applicable. Thus, the chase never introduces a violation of a functionality assertion. $\square$

We will now consider negative inclusion assertions. Ideally, we want to extend the previous lemma to negative inclusion assertions. But, it must not be the case that even if $DB\,(\mathcal{A})$ satisfies all NIs, $can\,(\mathcal{K})$ may not satisfy them. This is due to an interaction between NIs and PIs. Consider, for example, the PI $A_1 \sqsubseteq A_2$ and the NI $A_2 \sqsubseteq \neg A_3$. These two inclusion assertions logically imply the NI $A_1 \sqsubseteq \neg A_3$. An ABox $\mathcal{A}$ might satisfy both inclusion assertions separately but violates the implied NI. This interaction is captured by closing the negative inclusion assertions with respect to the positive inclusion assertions as defined next.

Definition 4.6. (Closure of NIs $cln\,(\mathcal{T})$ [23]) Let $\mathcal{T}$ be a *DL-Lite*$_\mathcal{A}$ TBox. We call *NI-closure* of $\mathcal{T}$, denoted by $cln\,(\mathcal{T})$, the TBox defined inductively as follows:

- all functionality assertions in $\mathcal{T}$ are also in $cln\,(\mathcal{T})$;
- all negative inclusion assertions in $\mathcal{T}$ are also in $cln\,(\mathcal{T})$;
- if $B_1 \sqsubseteq B_2$ is in $\mathcal{T}$ and $B_2 \sqsubseteq \neg B_3$ or $B_3 \sqsubseteq \neg B_2$ is in $cln\,(\mathcal{T})$, then also $B_1 \sqsubseteq \neg B_3$ is in $cln\,(\mathcal{T})$;
- if $Q_1 \sqsubseteq Q_2$ is in $\mathcal{T}$ and $\exists Q_2 \sqsubseteq \neg B$ or $B \sqsubseteq \neg \exists Q_2$ is in $cln\,(\mathcal{T})$, then also $\exists Q_1 \sqsubseteq \neg B$ is in $cln\,(\mathcal{T})$;
- if $Q_1 \sqsubseteq Q_2$ is in $\mathcal{T}$ and $\exists Q_2^- \sqsubseteq \neg B$ or $B \sqsubseteq \neg \exists Q_2^-$ is in $cln\,(\mathcal{T})$, then also $\exists Q_1^- \sqsubseteq \neg B$ is in $cln\,(\mathcal{T})$;
- if $Q_1 \sqsubseteq Q_2$ is in $\mathcal{T}$ and $Q_2 \sqsubseteq \neg Q_3$ or $Q_3 \sqsubseteq \neg Q_2$ is in $cln\,(\mathcal{T})$, then also $Q_1 \sqsubseteq \neg Q_3$ is in $cln\,(\mathcal{T})$;
- if one of the assertions $\exists Q \sqsubseteq \neg \exists Q$, $\exists Q^- \sqsubseteq \neg \exists Q^-$, or $Q \sqsubseteq \neg Q$ is in $cln\,(\mathcal{T})$, then all three such assertions are in $cln\,(\mathcal{T})$. $\triangleleft$

Notice that $cln\,(\mathcal{T})$ does not imply new negative inclusion assertions or functionality assertions not implied by $\mathcal{T}$. The next lemma, proven in [23], shows that we can use the ABox minimal model to check satisfiability of negative inclusions.

Lemma 4.3. *(Lemma 4.9 from [23]) Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ KB. Then, $can\left(\mathcal{K}\right)$ is a model of $\mathcal{K}$ if and only if $DB\left(\mathcal{A}\right)$ is a model of $\langle cln\left(\mathcal{T}\right), \mathcal{A}\rangle$.*

Now that we can check satisfiability of functionality and negative inclusion assertions we need to establish satisfiability of a *DL-Lite*$_\mathcal{A}$ KB. A *DL-Lite*$_\mathcal{A}$ KB $\mathcal{K}$ is satisfiable if $can\left(\mathcal{K}\right)$ is a model of $\mathcal{K}$ and vice versa (see Lemma 4.11 of [23]). Unfortunately, the construction of $can\left(\mathcal{K}\right)$ might not be convenient or even possible. If we combine the previous observations, we can show that satisfiability of a KB can be checked by looking at $DB\left(\mathcal{A}\right)$. This is captured by the following theorem, which follows from the previous established observations and lemmas.

Theorem 4.1. *(Theorem 4.12 from [23]) Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ KB. Then, $\mathcal{K}$ is satisfiable if and only if $DB\left(\mathcal{A}\right)$ is a model of $\langle cln\left(\mathcal{T}\right), \mathcal{A}\rangle$.*

We can verify whether $DB\left(\mathcal{A}\right)$ is a model of $\langle cln\left(\mathcal{T}\right), \mathcal{A}\rangle$ by evaluating a suitable boolean UCQ with inequalities over $DB\left(A\right)$. We define such a boolean UCQ via a translation δ from the assertions in $cln\left(\mathcal{T}\right)$ as follows [23]:

$$\begin{aligned}
\delta\left((\text{funct } P)\right) &= \exists x, y_1, y_2 . P\left(x, y_1\right) \wedge P\left(x, y_2\right) \wedge y_1 \neq y_2 \\
\delta\left(\left(\text{funct } P^-\right)\right) &= \exists x_1, x_2, y . P\left(x_1, y\right) \wedge P\left(x_2, y\right) \wedge x_1 \neq x_2 \\
\delta\left(B_1 \sqsubseteq \neg B_2\right) &= \exists x . \gamma_1\left(B_1, x\right) \wedge \gamma_2\left(B_2, x\right) \\
\delta\left(Q_1 \sqsubseteq \neg Q_2\right) &= \exists x, y . Q_1\left(x, y\right) \wedge Q_2\left(x, y\right)
\end{aligned}$$

where in the last two equations

$$\gamma_i\left(B, x\right) = \begin{cases} A\left(x\right), & \text{if } B = A, \\ \exists y_i . P\left(x, y_i\right), & \text{if } B = \exists P, \\ \exists y_i . P\left(y_i, x\right), & \text{if } B = \exists P^-, \end{cases}$$

Notice that the queries ask for a violation of an assertion. Therefore, if the evaluation of the UCQ

$$\bigcup_{\alpha \in cln(\mathcal{T})} \delta(\alpha)$$

over $DB\left(\mathcal{A}\right)$ returns the empty set then $\mathcal{K}$ is satisfiable (see Lemma 4.13 of [23]). Hence, KB satisfiability in *DL-Lite*$_\mathcal{A}$ can be reduced to query evaluation over a database.

4.1.3 Universal Model

In the previous section we have established the notion of the *DL-Lite*$_\mathcal{A}$ chase and as a result we have defined the canonical interpretation $can\left(\mathcal{K}\right)$. The question is, whether such an interpretation is a representation of all possible interpretations of $\mathcal{K}$. Such an interpretation is called a *universal model*. We will show that $can\left(\mathcal{K}\right)$ is indeed a universal model. But first, we need to define universal models. These are defined in terms of homomorphisms as follows.

Definition 4.7. Let $\mathcal{K}$ be a *DL-Lite*$_{\mathcal{A}}$ KBs; $\mathcal{I}$ and $\mathcal{J}$ be interpretations of the KB.

1. A *homomorphisms* $h : \mathcal{I} \to J$ is a mapping from $\Delta^{\mathcal{I}}$ to $\Delta^{\mathcal{J}}$ such that:

   (i) for all $a \in \Delta^{\mathcal{I}}$ and for each atomic concept $C \in \mathit{concepts}\,(\mathcal{K})$: if $a \in C^{\mathcal{I}}$ then $h\,(a) \in C^{\mathcal{J}}$,

   (ii) for all $(a, b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{J}}$ and for each atomic role $P \in \mathit{roles}\,(\mathcal{K})$: if $(a, b) \in P^{\mathcal{I}}$ then $(h\,(a)\,, h\,(b)) \in P^{\mathcal{J}}$.

2. $\mathcal{I}$ is *homomorphically equivalent* to $\mathcal{J}$ if there is a homomorphism $h : \mathcal{I} \to \mathcal{J}$ and a homomorphism $h' : \mathcal{J} \to \mathcal{I}$. ◁

Definition 4.8. (Universal model [1,48]) Let $\mathcal{K}$ be a *DL-Lite*$_{\mathcal{A}}$ KB. A universal model for $\mathcal{K}$ is a model $\mathcal{U} \models \mathcal{K}$ such that for every model $\mathcal{I} \models \mathcal{K}$, there exists a homomorphism $h : \Delta^{\mathcal{U}} \to \Delta^{\mathcal{I}}$. ◁

With this definition we can show that $\mathit{can}\,(\mathcal{K})$ is indeed a universal model for $\mathcal{K}$.

Theorem 4.2. *(Theorem 4 of [48]) Let $\mathcal{K}$ be a DL-Lite$_{\mathcal{A}}$ KB. If $\mathcal{K}$ is satisfiable, then $\mathit{can}\,(\mathcal{K})$ is a universal model for $\mathcal{K}$.*

Proof idea. Since $\mathcal{K}$ is satisfiable, $\mathit{can}\,(\mathcal{K})$ is a model of $\mathcal{K}$. It remains to show that for any model $\mathcal{I}$ of $\mathcal{K}$ it holds that there exists a homomorphism from $\mathit{can}\,(\mathcal{K})$ to $\mathcal{I}$. It is easy to see, that for each model $\mathcal{I}$ of $\mathcal{K}$ it holds that there is a homomorphism h from $DB\,(\mathcal{A})$ to $\mathcal{I}$. As we extend the interpretation in each chase step, we can extend the homomorphism h in each chase step to the (possible) new introduced constants, resulting in a homomorphism h'. Finally, at the end of the chase, we have constructed a homomorphism from $\mathit{can}\,(\mathcal{K})$ to $\mathcal{I}$, which proves that $\mathit{can}\,(\mathcal{K})$ is a universal model of $\mathcal{K}$. □

4.1.4 Query Answering over finite interpretations

Query answering in *DL-Lite*$_{\mathcal{A}}$ is an important task. Several methods for query answering in *DL-Lite*$_{\mathcal{A}}$ KBs exist. It is possible to evaluate a query over the canonical model of the KB (see Theorem 6 and Corollary 3 of [48]). The drawback of this method is that the canonical model might be infinite. In this section we will show a syntactic criterion that ensures a finite chase. We will present sets of *weakly-acyclic positive inclusion assertions*, which are a DL version of *weakly-acyclic tuple-generating dependencies* introduced by Fagin et al. [41]. We can show that any chase of a KB with weakly-acyclic PIs is polynomial in the size of the KB.

Let $\mathcal{K}$ be a *DL-Lite*$_{\mathcal{A}}$ KB. We denote with $B_{\mathcal{K}}$ the set of all basic concepts occurring in $\mathcal{K}$. The set $B_{\mathcal{K}}$ consists of atomic concepts and concepts of the form $\exists R$ or $\exists R^{-}$, where are R is an atomic role. A *dependency graph* is defined as follows.

Definition 4.9. (Dependency graph [48]) Let $\mathcal{K}$ be a *DL-Lite*$_{\mathcal{A}}$ KB. A dependency graph for $\mathcal{K}$, denoted as $G_{\mathcal{K}}$, is a directed edge-labeled graph, such that:

1. the set of nodes of $G_{\mathcal{K}}$ is $B_{\mathcal{K}}$;

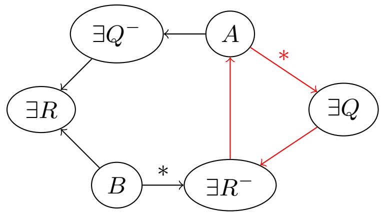


Figure 4.2: A dependency graph $G_{\mathcal{K}'}$ with a cycle (in red) that contains a $*$-labeled edge

2. the set of non-labeled edges of $G_{\mathcal{K}}$ is defined as follows:

   (a) For every PI $B \sqsubseteq B'$ in $\mathcal{K}$, where B and B' are basic concepts, there is a non-labeled edge from B to B', denoted by $B \longrightarrow B'$.

   (b) For every PI $Q_1 \sqsubseteq Q_2$ in $\mathcal{K}$, where Q_1 and Q_2 are basic roles, there are non-labeled edges $\exists Q_1 \longrightarrow \exists Q_2$ and $\exists Q_1^- \longrightarrow \exists Q_2^-$.

3. the set of $*$-labeled edges of $B_{\mathcal{K}}$ is defined as follows: Let B and B' be two nodes in $G_{\mathcal{K}}$. If it holds that

   (a) there is an edge $B \longrightarrow B'$,

   (b) $B' = \exists R$ or $B' = \exists R'$, and

   (c) $\exists R^- \in B_{\mathcal{K}}$ or $\exists R \in B_{\mathcal{K}}$ respectively,

   then there is a $*$-labeled edge $B \longrightarrow^* \exists R^-$ or $B \longrightarrow^* \exists R$ in $G_{\mathcal{K}}$. ◁

Definition 4.10. (Weak-acyclicity) A set of PIs is *weakly-acyclic* if its dependency graph has no cycles that contain a $*$-labeled edge. A KB is weakly-acyclic if the set of all its inclusion assertions is weakly-acyclic. ◁

Example 4.6. Let us consider the *DL-Lite*$_{\mathcal{A}}$ KB $\mathcal{K}'$ consisting of the following set of PIs:

$$B \sqsubseteq \exists R \qquad A \sqsubseteq \exists Q^- \qquad \exists R^- \sqsubseteq A \qquad Q^- \sqsubseteq R$$

The dependency graph consists of the nodes $B_{\mathcal{K}} = \{A, B, \exists R, \exists R^-, \exists Q, \exists Q^-\}$ and is depicted in Figure 4.2.
Since the dependency graph has a cycle that contains a $*$-labeled edge, the KB $\mathcal{K}'$ is not weakly-acyclic. ◁

The intuition of a dependency graph is that each non-labeled edge keeps track of the fact that a constant may be propagated during the chase from the concept at the origin to the concept at the end of the edge. Edges labeled with a $*$ keep track of newly introduced constants. If now a cycle goes through a labeled edge, then the newly introduced constant introduces again another new constant at a later chase step. Therefore, the chase continues forever and leads to an infinite interpretation. It is possible to show the following:

Theorem 4.3. *(Theorem 7 in [48]) For a satisfiable weakly-acyclic DL-Lite$_\mathcal{A}$ KB its chase has depth which is polynomial in the size of the KB.*

Proof. The proof is similar to the proof of Theorem 3.9 in [41]. □

It immediately follows that for a satisfiable weakly-acyclic *DL-Lite*$_\mathcal{A}$ KB, we can compute the chase and evaluate any query over the canonical model.

4.1.5 Query Answering over infinite interpretations

A more general method for query answering was introduced in [23]. This method separates the intensional and extensional level of the *DL-Lite*$_\mathcal{A}$ KB. First, the query is processed and reformulated based on the assertions in the TBox. Then, the reformulated query is evaluated over the ABox. This is similar to the presented method for KB satisfiability.

4.1.5.1 Query Reformulation

We are going to present the query reformulation algorithm as introduced in [23]. First, we define some preliminary notions. We distinguish between *bound* and *unbound* arguments of an atom in a query. Bound variables correspond to distinguished or shared variables, which are variables that either occur at least twice in the queries body or are a constant. The other variables are called unbound. The symbol '_' is used to represent non-distinguished non-shared, i.e. unbound, variables.

Next we define when a PI is applicable to an atom g [23]:

- A PI α is applicable to an atom $A(x)$, if α has A in its right-hand side.
- A PI α is applicable to an atom $P(x_1, x_2)$, if one of the following conditions holds:
  - (i) $x_2 = \_$ and the right-hand side of α is $\exists P$; or
  - (ii) $x_1 = \_$ and the right-hand side of α is $\exists P^-$; or
  - (iii) α is a role inclusion assertion and its right-hand side is either P or P^-.

The function $gr(g, \alpha)$ returns the atom obtained from g by applying the applicable inclusion α as defined by the following table:

Atom g	Positive inclusion α	$gr(g,\alpha)$
$A(x)$	$A_1 \sqsubseteq A$	$A_1(x)$
$A(x)$	$\exists Q \sqsubseteq A$	$Q(x, \_)$
$Q(x, \_)$	$A \sqsubseteq \exists Q$	$A(x)$
$Q(x, \_)$	$\exists Q_1 \sqsubseteq \exists Q$	$Q_1(x, \_)$
$Q(x_1, x_2)$	$Q_1 \sqsubseteq Q$	$Q_1(x_1, x_2)$

Table 4.1: The result $gr(g, \alpha)$ of applying a positive inclusion α to an atom g (Fig. 12 adapted from [23])

input : UCQ q, *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}$
output: UCQ pr

```
1  pr ← q;
2  repeat
3  |  pr' ← pr;
4  |  foreach CQ q' ∈ pr' do
5  |  |  foreach atom g in q' do
6  |  |  |  foreach PI α in T do
7  |  |  |  |  if α is applicable to g then
8  |  |  |  |  |  pr ← pr ∪ {q' [g/gr (g, α)]};
9  |  |  |  |  end
10 |  |  |  end
11 |  |  end
12 |  |  foreach pair of atoms g1, g2 in q' do
13 |  |  |  if g1 and g2 unify then
14 |  |  |  |  pr ← pr ∪ {anon (reduce (q', g1, g2))};
15 |  |  |  end
16 |  |  end
17 |  end
18 until pr' = pr;
19 return pr
```

Algorithm 4.1: The algorithm PerfectRef that computes the perfect reformulation of a CQ w.r.t. a *DL-Lite*$_{\mathcal{A}}$ TBox (Fig. 13 from [23])

The algorithm PerfectRef, given as Algorithm 4.1, reformulates a UCQ by taking into account the PIs of a TBox $\mathcal{T}$. As a first step (line 5 of Algorithm 4.1), the algorithm reformulates the atoms of each CQ $g' \in q'$, and produces a new query for each atom reformulation. This is denoted by $q'\left[g/gr\left(g, \alpha\right)\right]$, which means that we replace in q' each atom g with a new atom g', obtained by $gr\left(g, \alpha\right)$. As a second step (line 12 of Algorithm 4.1), we look for pairs of atoms g_1 and g_2 that unify. The function *reduce* then returns a new CQ by applying the most general unifier to the atoms g_1 and g_2. Therefore, variables that are bound may become unbound in the new CQ. The function *anon* then replaces each unbound variable by the symbol '_'.
Notice that the reformulation only depends on the PIs of a *DL-Lite*$_{\mathcal{A}}$ TBox. Actually it is the case that once we have established KB satisfiability, we can discard NIs and functionality assertions for query answering. It has been shown that the algorithm PerfectRef terminates [23].

Example 4.7. Consider the *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}_c$ given in Example 4.2. Now consider the CQ q over $\mathcal{T}_c$:

$$q(x) \leftarrow located\left(x, y\right), has\_room\left(\_, y\right),$$

which queries for courses that are located in a room that belongs to a building. We will go through the steps of the algorithm PerfectRef$(\{q\}, \mathcal{T}_c)$. In the first iteration the algorithm applies to the atom $has\_room\,(\_, y)$ the PI $room \sqsubseteq \exists has\_room^-$ and adds to pr the new query:

$$q(x) \leftarrow \mathit{located}\,(x, y)\,, \mathit{room}\,(y)\,.$$

In the next iteration, the PI $\exists located^- \sqsubseteq room$ is applied to the atom $room(y)$ and the following query is inserted in pr:

$$q(x) \leftarrow \mathit{located}\,(x, y)\,, \mathit{located}\,(\_, y)\,.$$

Notice that there are now two atoms $\mathit{located}\,(x, y)$ and $\mathit{located}\,(\_, y)$ that can be unified. The function $\mathit{reduce}\,(q, \mathit{located}\,(x, y)\,, \mathit{located}\,(\_, y))$ returns the atom $\mathit{located}\,(x, y)$. Since y is unbound, the function $anon$ replaces y by _. Therefore, the algorithm inserts in pr the new query:

$$q(x) \leftarrow \mathit{located}\,(x, \_)\,.$$

In the next iteration, it is, due to unification, possible to apply the PI $course \sqsubseteq \exists located$ to $\mathit{located}\,(x, \_)$. This inserts in pr the new query

$$q(x) \leftarrow \mathit{course}\,(x)\,.$$

At a further iteration, the algorithm applies the PI $\exists is\_of\_type \sqsubseteq course$ to $course(x)$ and adds to pr the new query

$$q(x) \leftarrow \mathit{is\_of\_type}\,(x, \_)\,.$$

Finally, the set of the five queries from above and the original query is returned by the algorithm $\mathsf{PerfectRef}(\{q\}, \mathcal{T}_c)$. ◁

Given that a *DL-Lite*$_\mathcal{A}$ KB $\mathcal{K}$ is satisfiable, the query returned by PerfectRef can be evaluated over the ABox $\mathcal{A}$ considered as a relational database, denoted by $DB\,(\mathcal{A})$. The returned answers are correct and coincide with the certain answers $\mathit{cert}\,(q, \mathcal{K})$ (see Theorem 5.14 of [23]). The query returned by $\mathsf{PerfectRef}(\{q\}, \mathcal{T})$ is called the *perfect rewriting* of q.

4.1.6 Additional Notions

In this section we further need the notion of isomorphisms and bisimulations as defined as follows. Isomorphisms and bisimulations help us to establish a relationship between two structures, which are in our case *DL-Lite*$_\mathcal{A}$ interpretations.

Definition 4.11. (Isomorphism) Let $\mathcal{I}$ and $\mathcal{J}$ be two models of a *DL-Lite*$_\mathcal{A}$ KB. $\mathcal{I}$ and $\mathcal{J}$ are *isomorphic* denoted as $\mathcal{I} \cong \mathcal{J}$ if there exists a bijective function $h : \Delta^\mathcal{I} \to \Delta^\mathcal{J}$, s.t.

- for all $a \in \Delta^\mathcal{I}$ and for each atomic concept $C \in \mathit{concepts}(\mathcal{T})$: $a \in C^\mathcal{I}$ if and only if $\left(a^\mathcal{I}\right) \in C^\mathcal{J}$, and
- for all $(a, b) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ and for each atomic role $R \in \mathit{roles}(\mathcal{T})$: $\left(a^\mathcal{I}, b^\mathcal{I}\right) \in R^\mathcal{I}$ if and only if $\left(h\left(a^\mathcal{I}\right), h\left(b^\mathcal{I}\right)\right) \in P^\mathcal{J}$. ◁

Notice that an isomorphism is a bijective homomorphism.

Definition 4.12. (Bisimulation (adapted to *DL-Lite*$_\mathcal{A}$ from [62])) A bisimulation $\sim_\mathcal{B}$ between two *DL-Lite*$_\mathcal{A}$ interpretations $\mathcal{I}$ and $\mathcal{J}$ is a relation in $\Delta^\mathcal{I} \times \Delta^\mathcal{J}$ such that, for every pair of objects $o_1 \in \Delta^\mathcal{I}$ and $o_2 \in \Delta^\mathcal{J}$, if $o_1 \sim_\mathcal{B} o_2$ then the following hold:

- for every atomic concept A: $o_1 \in A^\mathcal{I}$ if and only if $o_2 \in A^\mathcal{J}$;
- for every atomic role P:
  - for each o_1' with $(o_1, o_1') \in P^\mathcal{I}$, there is an o_2' with $(o_2, o_2') \in P^\mathcal{J}$ such that $o_1' \sim_\mathcal{B} o_2'$;
  - for each o_2' with $(o_2, o_2') \in P^\mathcal{J}$, there is an o_1' with $(o_1, o_1') \in P^\mathcal{I}$ such that $o_1' \sim_\mathcal{B} o_2'$;
- for every atomic role P (inverse property):
  - for each o_1' with $(o_1', o_1) \in P^\mathcal{I}$, there is an o_2' with $(o_2', o_2) \in P^\mathcal{J}$ such that $o_1' \sim_\mathcal{B} o_2'$;
  - for each o_2' with $(o_2', o_2) \in P^\mathcal{J}$, there is an o_1' with $(o_1', o_1) \in P^\mathcal{I}$ such that $o_1' \sim_\mathcal{B} o_2'$; $\lhd$

Let π, π' be two sets of objects, such that $\pi \subseteq \Delta^\mathcal{I}$ and $\pi' \subseteq \Delta^\mathcal{J}$. We say π bisimulates π', denoted by $\pi \sim_\mathcal{B} \pi'$, if every object $o \in \pi$ bisimulates every object in π' and vice versa. That is, for every $o \in \pi$, it holds that $o \sim_\mathcal{B} o'$ for all $o' \in \pi'$ and for every $o' \in \pi'$, it holds that $o \sim_\mathcal{B} o'$ for all $o \in \pi$.

4.2 A Direct Mapping of Relational Data to Description Logic Knowledge Bases

In this section we will show how to map relational data to *DL-Lite*$_\mathcal{A}$ KBs. First, we will review the direct mapping of relational data to RDF as presented by the W3C [5] and Sequeda et al. [63]. We will discuss their weaknesses regarding the lack of means to capture all semantic information of the relational model. We will therefore introduce a direct mapping of relational data to *DL-Lite*$_\mathcal{A}$ knowledge bases that overcomes the limitations of the previous translations.

4.2.1 A Direct Mapping of Relational Data to RDF

Most of the data in information systems is stored in relational databases. It is important for the success of the Semantic Web to utilize the information stored in relational databases. Therefore an automatic translation of relational data to RDF is needed. The W3C consortium introduced such a translation, called “A Direct Mapping of Relational Data to RDF” (RDB-direct-mapping) [5]. This direct mapping defines an RDF graph representation of the data in a relational database. Such an RDF graph can then be further processed, for example it can be queried using an RDF query language or it can be merged with other RDF graphs to add further information to the information specified in a relational table.

The RDB-direct-mapping specifies that a relational database is translated into an RDF graph, which is called *direct graph*. Such a direct graph is the union of the *table graphs* for each table in a relational schema. A table graph is the union of *row graphs* for each row in a table. A row

$$(\texttt{course/lecture=AlgebraI;type=VO}, \texttt{rdf:type}, \textit{course}) \tag{4.1}$$
$$(\texttt{course/lecture=AlgebraI;type=VO}, \texttt{course\#ref-room}, \texttt{rooms/room=HS1}) \tag{4.2}$$
$$(\texttt{course/lecture=AlgebraI;type=VO}, \texttt{course\#lecture}, \text{“Algebra I”}) \tag{4.3}$$
$$(\texttt{course/lecture=AlgebraI;type=VO}, \texttt{course\#type}, \text{“VO”}) \tag{4.4}$$
$$(\texttt{course/lecture=AlgebraI;type=VO}, \texttt{course\#room}, \text{“HS1”}) \tag{4.5}$$
$$(\texttt{course/lecture=AlgebraI;type=UE}, \texttt{rdf:type}, \textit{course}) \tag{4.6}$$
$$(\texttt{course/lecture=AlgebraI;type=UE}, \texttt{course\#ref-room}, \texttt{rooms/room=SEM1}) \tag{4.7}$$
$$(\texttt{course/lecture=AlgebraI;type=UE}, \texttt{course\#lecture}, \text{“Algebra I”}) \tag{4.8}$$
$$(\texttt{course/lecture=AlgebraI;type=UE}, \texttt{course\#type}, \text{“UE”}) \tag{4.9}$$
$$(\texttt{course/lecture=AlgebraI;type=UE}, \texttt{course\#room}, \text{“SEM1”}) \tag{4.10}$$
$$(\texttt{course/lecture=EconomicsI;type=UE}, \texttt{rdf:type}, \textit{course}) \tag{4.11}$$
$$(\texttt{course/lecture=EconomicsI;type=UE}, \texttt{course\#ref-room}, \texttt{rooms/room=SEM1}) \tag{4.12}$$
$$(\texttt{course/lecture=EconomicsI;type=UE}, \texttt{course\#lecture}, \text{“Economics I”}) \tag{4.13}$$
$$(\texttt{course/lecture=EconomicsI;type=UE}, \texttt{course\#type}, \text{“UE”}) \tag{4.14}$$
$$(\texttt{course/lecture=EconomicsI;type=UE}, \texttt{course\#room}, \text{“SEM1”}) \tag{4.15}$$

Figure 4.3: The RDF triples obtained by the RDB-direct-mapping [5] from the relational table in Figure 2.2. In red are the triples that are in the result of the RDB-direct-mapping but not in the result of the direct mapping $\mathcal{DM}$ [63], which will be introduced next.

graph is an RDF representation of a row of a relational table. Each row is identified by a *row node*. The row node has as name, its table's name and the values of all primary key columns, for example `course/lecture=AlgebraI;type=VO`. If a table does not have a primary key, then a fresh blank node is used as a row node. For each row of a table, the row graph consists of the following:

- A *row type triple* which specifies the type (table) of a row node (see number 4.1, 4.6 and 4.11 in Figure 4.3).
- *Reference triples* represent the foreign keys of a table. For this a special predicate is used. This predicate's label consists of the tables name, the keyword `ref` and the columns name, for example `course#ref-room` (see numbers 4.2, 4.7 and 4.12 in Figure 4.3).
- *Literal triples* store the value of the row's columns. The predicate that is used for literal triples contains the table's name and the column's name, for example `course#room` (see numbers 4.3-4.5, 4.8-4.10 and 4.13-4.15 in Figure 4.3).

The RDB-direct-mapping does not add any further information on the semantics of the data in the relational database. This allows to specify information that cannot be present in any relational database. Consider for example the triple

$$(\texttt{course/lecture=AlgebraI;type=VO}, \texttt{rdf:type}, \textit{rooms}),$$

which extends the RDF triples given in Figure 4.3. Such a triple would infer that the row, which is identified by the row node `course/lecture=AlgebraI;type=VO` is also a row in the "rooms" table. But, the relational model does not allow for rows that belong to different tables. Hence, the RDB-direct-mapping does not preserve the semantics of the relational model.

For a direct mapping $\mathcal{M}$ it would be desirable to preserve the semantics of the relational model. Additionally, a direct mapping should have other fundamental and desirable properties. These are introduced in [63] and are defined as follows:

Fundamental properties

- *Information preservation* guarantees that the mapping does not loose any information. Formally, a direct mapping is information preserving if we can define a computable function $\mathcal{N}$, such that $\mathcal{N}$ translates RDF graphs to instances of a relational schema R. Additionally, the result of $\mathcal{N}$, applied to an RDF graph directly mapped from an instance I of a relational schema, should return the same instance I, i.e. $\mathcal{N}(\mathcal{M}(R, I)) = I$.
- *Query preservation* guarantees that everything that can be extracted from a relational database, can also be extracted from the translated RDF graph via a suitable query language. That is, we can translate queries over the relational database into equivalent queries over RDF graphs.

Notice that neither of the two properties includes the other. Thus, we can have a direct mapping that preserves information, but maps this information such that a more powerful query language has to be used. On the other side, if, for example, the instances are mapped, but not the schema, it might be the case that the mapping is query preserving but not information preserving.

Desirable properties

- *Monotonicity*: Let I_1 and I_2 be instances of a relational schema, such that I_1 is contained in I_2. Then a direct mapping is monotone, if the direct mapping of I_2 is contained in I_1. Therefore, with a monotone direct mapping whenever we add new data to the relational database, we only have to map the new data and not the whole database.
- *Semantics preservation* guarantees that data dependencies are encoded in the translation process. Let I be an instance of a relational schema R and Σ a set of data dependencies. Then, a direct mapping $\mathcal{M}$ is called semantics preserving, if $I \vDash \Sigma$ if and only if $\mathcal{M}(R, I)$ is a model of the translated data dependencies.

The direct mapping $\mathcal{DM}$ introduced by Sequeda et al. [63] extends the RDB-direct-mapping in such a way that it is information and query preserving. $\mathcal{DM}$ translates the relational schema and the relational instances via Datalog rules into an RDF graph. This process is similar to the direct mapping introduced previously. We will present here the differences from the RDB-direct-mapping. Additionally, for the ease of presentation, we will ignore the fact that the direct mapping $\mathcal{DM}$ reverse engineers binary relations that result from modeling $n : m$-relationships of the conceptual model. Therefore, we will discuss the direct mapping from a database consisting of a single relation, and compare the result with the result from the RDB-direct-mapping depicted in Figure 4.3. The direct mapping $\mathcal{DM}$ translates a relational schema $R[A_1, \ldots, A_n]$ as follows:

Relational instances are translated as in the RDB-direct-mapping, except:

- Columns that are foreign keys directly refer to the row node of the referenced table. We do not store a tuple with a predicate having the "#ref" keyword. For example, in Figure 4.4 triples numbered 4.27, 4.31 and 4.35 are added, and the triples drawn in red of Figure 4.3 are not in the result of the translation.
- Binary relationships between two tables are directly modeled with a single predicate (not considered here).

Relational schema is translated as follows:

- The RDF representation of a table is of type `owl:Class` (see triple 4.16 in Figure 4.4).
- The columns of a table are represented as OWL properties. Columns that represent foreign keys are object properties and the others are datatype properties (see triples 4.17, 4.19 and 4.21 in Figure 4.4).
- The predicates are typed, i.e. we specify domain and range of each object property and domain of each datatype property (see triples 4.18, 4.20, 4.22 and 4.23 in Figure 4.4).

The direct mapping $\mathcal{DM}$ is information preserving, query preserving and monotone, but it is not semantics preserving [63]. Consider an instance consisting of two rows with the same primary key, but in the other columns the rows have different values. Such an instance is clearly inconsistent. The result of the direct mapping $\mathcal{DM}$ of such an instance is a consistent RDF graph. We have not yet mapped the restrictions enforced by primary and foreign keys. The problem is that OWL does not have a method to specify primary keys. One solution is to encode the violation of a primary key constraint into the mapping. This is done as follows. A datalog rule is defined, which detects a primary key constraint violation. Then, the tuple `a owl:differentFrom a` is added to the set of RDF triples. This tuple makes the RDF graph inconsistent. This new direct mapping is called $\mathcal{DM}_{pk}$. For this direct mapping the following proposition holds.

Proposition 4.1. *(Proposition 2 from [63]) The direct mapping $\mathcal{DM}_{pk}$ is information preserving, query preserving, monotone, and semantics preserving if one considers only primary keys. That is, for every relational schema R, set Σ of (only) primary keys over R and instance I of R: $I \vDash \Sigma$ iff $\mathcal{DM}_{pk}\,(R, \Sigma, I)$ is consistent under OWL semantics.*

The direct mapping $\mathcal{DM}_{pk}$ can be extended with the same idea also to consider foreign keys. Unfortunately, this extension leads to a non-monotone direct mapping. This is because a foreign key inconsistency can be repaired by adding new tuples. But then, the introduced triple `a`

$$(\textit{course}, \texttt{rdf:type}, \texttt{owl:Class}) \tag{4.16}$$
$$(\texttt{course\#lecture}, \texttt{rdf:type}, \texttt{owl:DatatypeProperty}) \tag{4.17}$$
$$(\texttt{course\#lecture}, \texttt{rdfs:domain}, \textit{course}) \tag{4.18}$$
$$(\texttt{course\#type}, \texttt{rdf:type}, \texttt{owl:DatatypeProperty}) \tag{4.19}$$
$$(\texttt{course\#type}, \texttt{rdfs:domain}, \textit{course}) \tag{4.20}$$
$$(\texttt{course\#room}, \texttt{rdf:type}, \texttt{owl:ObjectProperty}) \tag{4.21}$$
$$(\texttt{course\#room}, \texttt{rdfs:domain}, \textit{course}) \tag{4.22}$$
$$(\texttt{course\#room}, \texttt{rdfs:range}, \textit{rooms}) \tag{4.23}$$
$$(\texttt{course/lecture=AlgebraI;type=VO}, \texttt{rdf:type}, \textit{course}) \tag{4.24}$$
$$(\texttt{course/lecture=AlgebraI;type=VO}, \texttt{course\#lecture}, \text{“Algebra I”}) \tag{4.25}$$
$$(\texttt{course/lecture=AlgebraI;type=VO}, \texttt{course\#type}, \text{“VO”}) \tag{4.26}$$
$$(\texttt{course/lecture=AlgebraI;type=VO}, \texttt{course\#room}, \texttt{rooms/room=HS1}) \tag{4.27}$$
$$(\texttt{course/lecture=AlgebraI;type=UE}, \texttt{rdf:type}, \textit{course}) \tag{4.28}$$
$$(\texttt{course/lecture=AlgebraI;type=UE}, \texttt{course\#lecture}, \text{“Algebra I”}) \tag{4.29}$$
$$(\texttt{course/lecture=AlgebraI;type=UE}, \texttt{course\#type}, \text{“UE”}) \tag{4.30}$$
$$(\texttt{course/lecture=AlgebraI;type=UE}, \texttt{course\#room}, \texttt{rooms/room=SEM}) \tag{4.31}$$
$$(\texttt{course/lecture=EconomicsI;type=UE}, \texttt{rdf:type}, \textit{course}) \tag{4.32}$$
$$(\texttt{course/lecture=EconomicsI;type=UE}, \texttt{course\#lecture}, \text{“Economics I”}) \tag{4.33}$$
$$(\texttt{course/lecture=EconomicsI;type=UE}, \texttt{course\#type}, \text{“UE”}) \tag{4.34}$$
$$(\texttt{course/lecture=EconomicsI;type=UE}, \texttt{course\#room}, \texttt{rooms/room=SEM1}) \tag{4.35}$$

Figure 4.4: The RDF triples obtained by the direct mapping $\mathcal{DM}$ [63] from the relational table in Figure 2.2. In green are the triples that are in the result of the direct mapping $\mathcal{DM}$ but not in the RDB-direct-mapping.

`owl:differentFrom a` has to be removed from the RDF graph. Hence, the previous model is not a submodel of the new model, thus violating monotonicity.

4.2.2 A Direct Mapping of Relational Data to *DL-Lite*$_{\mathcal{A}}$ Knowledge Bases

The direct mapping introduced in the previous section lays the foundations of the mapping we will introduce in this section. This direct mapping translates relational schemas and instances to *DL-Lite*$_{\mathcal{A}}$ KBs. We do not consider primary and foreign keys as data dependencies, we rather focus on functional dependencies. We will show that our mapping is semantics preserving with respect to functional dependencies. First, we will define a mapping from a relational schema to a *DL-Lite*$_{\mathcal{A}}$ KB, and will later extend this definition with a mapping from functional dependencies to dependencies for *DL-Lite*$_{\mathcal{A}}$. We will call the combination of both mappings *relational to Description Logic direct mapping* (R2DM).

In the following, let $R\,[A_1, \ldots, A_n]$ be a relational schema. For the mapping we will use concepts R, R_A_i and roles $R\#A_i$. Tuples occurring in database instances of R are represented by instances of R concepts, a value in a column is represented by an instance of an R_A_i concept, and the value is then specified by the $value$ value-domain. Without loss of generality, we assume that each column is of the domain `xsd:string`. The following definition uses dependencies for KBs in form of identification constraints (IdCs) to be defined in the later Section 4.3. The semantics of the particular IdC is given directly in the definition.

Definition 4.13. (Schema to DL direct mapping (sm)) Let $R[U]$ be a relational schema with attributes $U = A_1, \ldots, A_n$. The function $sm\,(R[U])$ outputs a *DL-Lite*$_\mathcal{A}$ T-Box $\mathcal{T}_{R[U]}$ with an IdC $\sigma_{R[U]}$ as follows:

- The first set of assertions ensures that the all concepts are disjoint from each other, e.g. an attribute cannot also denote a tuple:

$$\begin{aligned} R &\sqsubseteq \neg R\_A_i && \forall 1 \leq i \leq n \\ R\_A_i &\sqsubseteq \neg R\_A_j && \forall 1 \leq i < j \leq n \end{aligned}$$

- Next, we add assertions that express domain and range of a role, mandatory participation to a role as well as domain of an attribute:

$$\begin{aligned} \exists R\#A_i &\sqsubseteq R & \exists R\#A_i^- &\sqsubseteq R\_A_i & &\forall 0 < i \leq n \\ R &\sqsubseteq \exists R\#A_i & R\_A_i &\sqsubseteq \exists R\#A_i^- & &\forall 0 < i \leq n \\ R\_A_i &\sqsubseteq \delta\,(value) & & & &\forall 0 < i \leq n \\ R &\sqsubseteq \neg\delta\,(value) & \rho\,(value) &\sqsubseteq \texttt{xsd:string} & & \end{aligned}$$

- Last, we add functionality assertions to express that each tuple can only have one of each attribute:

$$\begin{aligned} &(\text{funct } R\#A_i) && \forall 0 < i \leq n \\ &(\text{funct } value) && \end{aligned}$$

- Additionally, since in the relational model set semantics is assumed, i.e. no tuple can occur twice, we add an identification constraint:

$$\sigma_{R[U]} = (\text{id} \;\; R \;\; R\#A_1, \ldots, R\#A_n)\,.$$

  This IdC says, that for any two instances of the R concept, if they agree on the instances connected by the $R\#A_1, \ldots, R\#A_n$ roles, then these two instances must be the same.

The function $sm\,(R[U])$ outputs $\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle$. $\triangleleft$

Example 4.8. We will now translate the relational schema $course(lecture, type, room)$ given in Example 2.2 to a *DL-Lite*$_\mathcal{A}$ KB using the schema-direct mapping, i.e. $sm\,(course(lecture, type, room))$ outputs the following *DL-Lite*$_\mathcal{A}$ TBox:

- Concept disjointness assertions:

$$\begin{aligned} \mathit{course} &\sqsubseteq \neg \mathit{course\_lecture} & \mathit{course} &\sqsubseteq \neg \mathit{course\_type} \\ \mathit{course} &\sqsubseteq \neg \mathit{course\_room} & \mathit{course\_lecture} &\sqsubseteq \neg \mathit{course\_type} \\ \mathit{course\_lecture} &\sqsubseteq \neg \mathit{course\_room} & \mathit{course\_type} &\sqsubseteq \neg \mathit{course\_room} \end{aligned}$$

- Role and attribute property assertions:

$$\begin{aligned} \exists \mathit{course\#lecture} &\sqsubseteq \mathit{course} & \exists \mathit{course\#lecture}^- &\sqsubseteq \mathit{course\_lecture} \\ \exists \mathit{course\#type} &\sqsubseteq \mathit{course} & \exists \mathit{course\#type}^- &\sqsubseteq \mathit{course\_type} \\ \exists \mathit{course\#room} &\sqsubseteq \mathit{course} & \exists \mathit{course\#room}^- &\sqsubseteq \mathit{course\_room} \\ \mathit{course} &\sqsubseteq \exists \mathit{course\#lecture} & \mathit{course\_lecture} &\sqsubseteq \exists \mathit{course\#lecture}^- \\ \mathit{course} &\sqsubseteq \exists \mathit{course\#type} & \mathit{course\_type} &\sqsubseteq \exists \mathit{course\#type}^- \\ \mathit{course} &\sqsubseteq \exists \mathit{course\#room} & \mathit{course\_room} &\sqsubseteq \exists \mathit{course\#room}^- \\ \mathit{course\_lecture} &\sqsubseteq \delta\,(\mathit{value}) & & \\ \mathit{course\_type} &\sqsubseteq \delta\,(\mathit{value}) & & \\ \mathit{course\_room} &\sqsubseteq \delta\,(\mathit{value}) & & \\ \mathit{course} &\sqsubseteq \neg\delta\,(\mathit{value}) & \rho\,(\mathit{value}) &\sqsubseteq \texttt{xsd:string} \end{aligned}$$

- Functionality assertions:

$$\begin{aligned} &(\text{funct } \mathit{course\#lecture}) \\ &(\text{funct } \mathit{course\#type}) \\ &(\text{funct } \mathit{course\#room}) \\ &(\text{funct } \mathit{value}) \end{aligned}$$

- Identification constraint:

$$(\text{id } \ \mathit{course} \ \ \mathit{course\#lecture}, \mathit{course\#type}, \mathit{course\#room}) \qquad \triangleleft$$

The function sm maps relational schemas to *DL-Lite*$_\mathcal{A}$ T-Boxes. Given such a mapping we can translate each model of the KB into an instance of the relational schema and vice versa. We now define such translations. Ideas for the translation and the upcoming proof were taken from [19]. The function $i2m_{R[U]}\,(I)$ translates an instance I of a relational schema $R[U]$ into a model of the KB created by the function sm.

Definition 4.14. (Instance to model mapping ($i2m$)) Let $I = (\mathbf{dom}^I, T^I)$ be an instance of $R[U]$. Let $\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle$ denote the result of the function $sm\,(R[U])$. Then, we build the interpretation $\mathcal{J} = (\Delta^\mathcal{J}, \cdot^\mathcal{J})$ of $\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle$ as follows:

- $\Delta^\mathcal{J} = \mathbf{dom}^I \cup \{c_{v,A_i} \mid A_i \in U \wedge v \in T^I\,[A_i]\} \cup \{t_{\langle d_1,\ldots,d_n\rangle} \mid \langle d_1, \ldots, d_n\rangle \in T^I\}$, where c_{v,A_i} and $t_{\langle d_1,\ldots,d_n\rangle}$ are new elements not yet in $\mathbf{dom}^I$

- $R^{\mathcal{J}} = \{t_{\langle d_1,\ldots,d_n\rangle} \mid \langle d_1,\ldots,d_n\rangle \in T^I\}$,
- $R\_A_i^{\mathcal{J}} = \{c_{v,A_i} \mid v \in T^I[A_i]\}$ for all attributes A_i in the relation R,
- $value^{\mathcal{J}} = \{(c_{v,A_i}, v) \mid v \in T^I[A_i]\}$ for all attributes A_i in the relation R,
- $R\#A_i^{\mathcal{J}} = \{(t_{\langle d_1,\ldots,d_n\rangle}, c_{v,A_i}) \mid \langle d_1,\ldots,d_n\rangle \in T^I \wedge v = \langle d_1,\ldots,d_n\rangle[A_i]\}$ for all attributes A_i in the relation R.

The function $i2m_{R[U]}(I)$ returns $\mathcal{J}$. $\triangleleft$

Notice, that for each tuple we create new domain elements $t_{\langle d_1,\ldots,d_n\rangle}$, which uniquely identify each tuple in the returned model $\mathcal{J}$. We will call such domain elements $t_{\langle d_1,\ldots,d_n\rangle}$ *tuple identifiers*. For each value appearing in a column we create new domain elements c_{v,A_i}, which we will call *value identifiers*.

Example 4.9. We use the instance of Figure 2.2 and the relational schema of Example 2.2. We translate the relational schema into a TBox $\mathcal{T}_{R[U]}$ with the IdC $\sigma_{R[U]}$ according to the function sm (see Example 4.8). Now we will map the instance given in Figure 2.2 into an interpretation of $\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle$. This interpretation is depicted in Figure 4.5. We depict in red the tuple identifiers and in blue the value identifiers. Values are drawn in violet. $\triangleleft$

Lemma 4.4 shows that the model returned by $i2m_{R[U]}(\mathcal{I})$ is indeed a valid model of $\mathcal{T}_{R[U]}$ and the IdC $\sigma_{R[U]}$.

Lemma 4.4. *Let $R[U]$ be a relational schema, $\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle$ be the result of $sm(R[U])$, and I be an instance of $R[U]$. Let $\mathcal{J}$ be the interpretation returned by $i2m_{R[U]}(I)$. Then $\mathcal{J}$ is a model of $\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle$, i.e. $\mathcal{J} \models \mathcal{T}_{R[U]}$ and $\mathcal{J} \models \sigma_{R[U]}$.*

Proof. In order to show that $\mathcal{J}$ is a model of $\mathcal{T}_{R[U]}$ and $\sigma_{R[U]}$ we need to show that $\mathcal{J}$ satisfies all assertions in $\mathcal{T}_{R[U]}$ and the IdC $\sigma_{R[U]}$.

- By the definition of $sm(R[U])$ the following assertions appear in $\mathcal{T}_{R[U]}$:
  - $R \sqsubseteq \neg R\_A_i$: Since $R^{\mathcal{J}}$ contains only the domain elements not in **dom**, this assertion is satisfied.
  - $R\_A_i \sqsubseteq \neg R\_A_j$: For all $i \in [1 \ldots n]$, $R\_A_i^{\mathcal{J}}$ contains new domain elements not in any other $R\_A_j^{\mathcal{J}}$.
  - $\exists R\#A_i \sqsubseteq R$, $R \sqsubseteq \exists R\#A_i$: All tuples are in the interpretation of both concepts.
  - $\exists R\#A_i^- \sqsubseteq R\_A_i$, $R\_A_i \sqsubseteq \exists R\#A_i^-$: All attributes are in the interpretation of both concepts.
  - (funct $R\#A_i$): For each tuple we construct a single role interpretation.
  - (funct $value$): A column of a relational tuple cannot be occupied by two values.
- $\sigma_{R[U]}$ contains the following IdC:

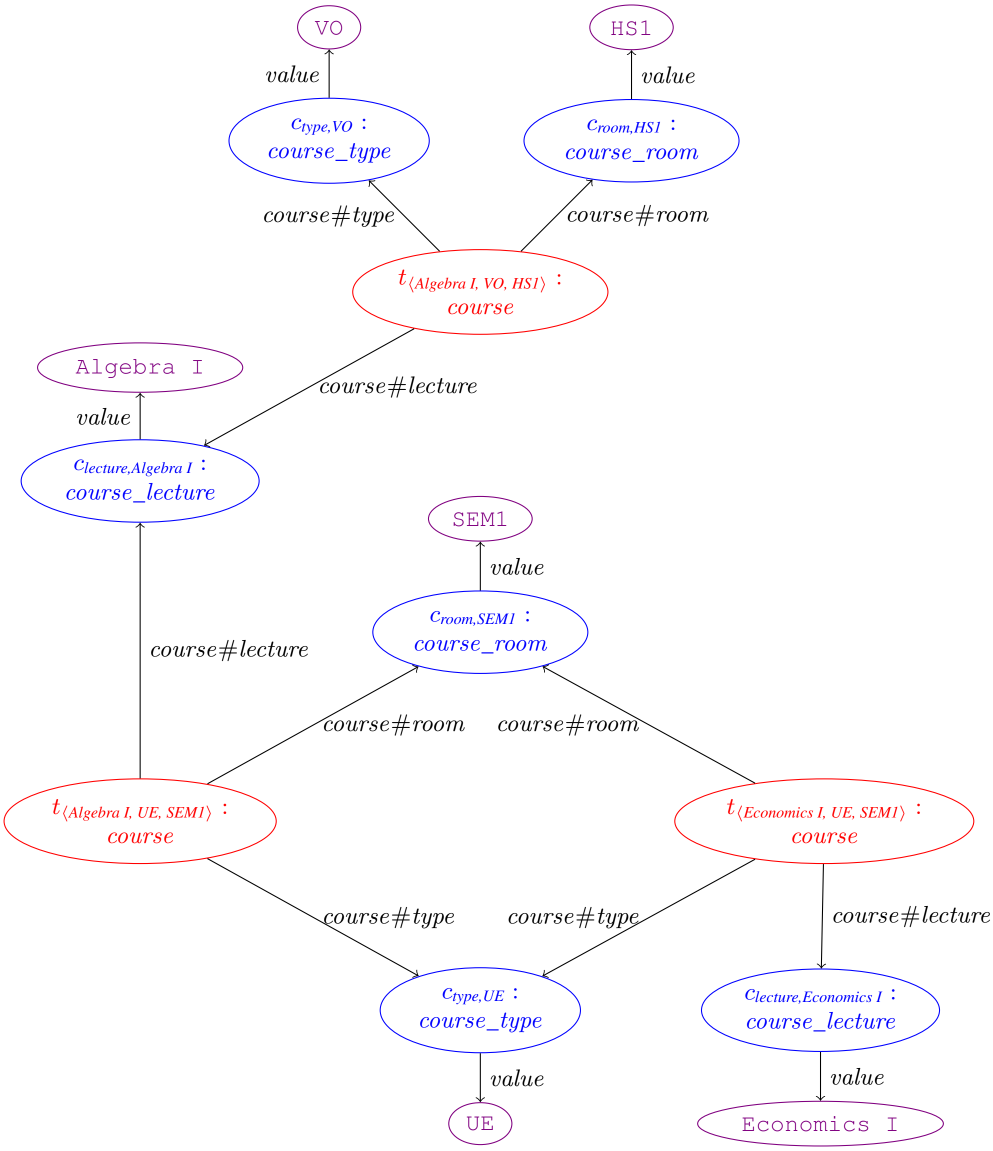


Figure 4.5: An interpretation translated by the function $i2m$ from the instance given in Figure 2.2.

– (id R $R\#A_1, \ldots, R\#A_n$): Since each tuple is unique, also the identification assertion is satisfied by $\mathcal{J}$.

Hence, all assertions in $\langle \mathcal{T}_{R[U]}, \Sigma \rangle$ are satisfied, i.e. $\mathcal{J}$ is a valid model of $\mathcal{T}_{R[U]}$. $\square$

The function $m2i_{\mathcal{T}_{R[U]}}(\mathcal{I})$ translates a model $\mathcal{I}$ of a KB created by $sm\,(R[U])$ into an instance of the relational schema $R[U]$.

Definition 4.15. (Model to instance mapping ($m2i$)) $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of $sm\,(R\,[U]) = \left\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \right\rangle$. Then we can build a pair $J = \left(\mathbf{dom}^J, T^J\right)$ of $R[U]$ as follows:

- $\mathbf{dom}^J = \Delta^{\mathcal{I}} \setminus \left(R^{\mathcal{I}} \cup \bigcup_{i=1}^{n} R_{A_i}^{\mathcal{I}}\right)$,
- $T^J = \{\langle v_0, \ldots, v_n\rangle \mid \exists t \in R^{\mathcal{I}} \text{ s.t. } \bigwedge_{i=0}^{n}(t, t_i) \in R\#A_i^{\mathcal{I}} \wedge \bigwedge_{i=0}^{n}(t_i, v_i) \in value^{\mathcal{I}}\}$.

The function $m2i_{\mathcal{T}_{R[U]}}\,(\mathcal{I})$ returns J. ◁

Lemma 4.5 shows that the pair $J = \left(\mathbf{dom}^J, T^J\right)$ returned by $m2i_{\mathcal{T}_{R[U]}}\,(\mathcal{J})$ is indeed an instance of $R[U]$.

Lemma 4.5. *Let $R[U]$ be a relational schema, $\left\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \right\rangle$ be the result of $sm\,(R[U])$, and $\mathcal{I}$ be a model of $\mathcal{T}_{R[U]}$. $J = \left(\mathbf{dom}^J, T^J\right)$ is the pair returned by $m2i_{\mathcal{T}_{R[U]}}\,(\mathcal{I})$. Then J is an instance of $R[U]$, i.e. $J \vDash R[U]$.*

Proof. We need to check if J is an instance of $R[U]$. Notice that T^J are the tuples of the relation R. Because of (id R $R\#A_1, \ldots, R\#A_n$) a tuple cannot be represented twice in $\mathcal{I}$, and therefore each tuple in T^J is unique. Because of the mandatory role participation assertions ($\exists R\#A_i^- \sqsubseteq R\_A_i$) we have at least one value for every attribute of each tuple. Since all roles $R\#A_i$ are functional, we have at most one value for every attribute of each tuple. Hence, each tuple of T^J is a tuple of $R[U]$. Therefore, J is an instance of $R[U]$. □

So far we have seen that $i2m$ and $m2i$ generate instances and models. The next lemma will show that the translation of instances to models and back to instances does not lose any information.

Lemma 4.6. *Let $R[U]$ be a relational schema and let $\left\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \right\rangle$ be the result of $sm\,(R[U])$. Let I be an instance of $R[U]$, then $I = m2i_{\mathcal{T}_{R[U]}}\left(i2m_{R[U]}\,(I)\right)$.*

Proof. By Lemma 4.4 $i2m_{R[U]}\,(I)$ outputs a model $\mathcal{J}$ of $\mathcal{T}_{R[U]}$. Additionally, each tuple of I is identified by a unique tuple identifier $t_{\langle d_1, \ldots, d_n\rangle}$ and no other tuple identifiers are generated. Each tuple identifier has n associated value identifiers, each connected to a single value. Since by Lemma 4.5, $m2i_{\mathcal{T}_{R[U]}}\,(\mathcal{J})$ outputs an instance which has exactly the same tuples as I. Therefore, $I = m2i_{\mathcal{T}_{R[U]}}\left(i2m_{R[U]}\,(I)\right)$. □

It is also possible to translate models to instances and back to models. But, we will then loose some information. During the translation from models to instances we drop the domain elements which represent tuple and value identifiers. Then, when we translate the instance back to a model, we generate new domain elements for tuple and value identifiers. Those are not necessarily the same as before. But, it is possible to map to each tuple and value identifier from the original model an identifier of the new model. Thus, we can show in the next lemma that the two models are isomorphic.

Lemma 4.7. *Let $R[U]$ be a relational schema and let $\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle$ be the result of $sm\,(R[U])$. Let $\mathcal{I}$ be a model of $\mathcal{T}_{R[U]}$, then $\mathcal{I} \cong i2m_{\mathcal{T}_{R[U]}}\left(m2i_{R[U]}\,(\mathcal{I})\right)$.*

Proof. By Lemma 4.5 $m2i_{R[U]}\,(\mathcal{I})$ outputs an instance J of $R[U]$. The function $i2m_{\mathcal{T}_{R[U]}}\,(J)$ then outputs a model $\mathcal{I}'$. In order to show that $\mathcal{I} \cong i2m_{\mathcal{T}_{R[U]}}\left(m2i_{R[U]}\,(\mathcal{I})\right)$, we are going to construct an isomorphism between $\mathcal{I}$ and $\mathcal{I}'$, therefore we define a bijective function $h : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}'}$. Since, $m2i$ and $i2m$ only change the tuple and value identifiers we map all other domain element to themselves:

- for all $d \in \Delta^{\mathcal{I}} \setminus \{R^{\mathcal{I}} \cup \bigcup_{i=1}^{n} R_{A_i}^{\mathcal{I}}\}$: $h\,(d) = d$

It remains to map the tuple and value identifiers, which are the domain elements in $R^{\mathcal{I}}$ and $R\_A_i^{\mathcal{I}}$. Since $\mathcal{I}$ is a valid model of $\mathcal{T}_{R[U]}$, each $t \in R^{\mathcal{I}}$ is connected by an $R\#A_i$ role to an R_A_i concept c_i. These c_i objects are connected to values d_i, which are used in $i2m$ to generate the tuple identifier $t_{\langle d_1,\ldots,d_n \rangle}$. Therefore, we map each $t \in R^{\mathcal{I}}$ to the corresponding tuple identifier $t_{\langle d_1,\ldots,d_n \rangle}$, and each $c_i \in R\_A_i^{\mathcal{I}}$ to the corresponding value identifier.

- for all $A_i \in U$ and for all $c_i \in R\_A_i^{\mathcal{I}}$: $h\,(c_i) = {c_{v,A_i}}^{\mathcal{I}'}$, such that $(c_i, v) \in value^{\mathcal{I}}$,
- for all $t \in R^{\mathcal{I}}$: $h\,(t) = {t_{\langle d_1,\ldots,d_n \rangle}}^{\mathcal{I}'}$, such that for all d_i it holds that $(t, c_i)^{\mathcal{I}} \in R\#A_i$ and $(c_i, d_i) \in value^{\mathcal{I}}$.

Since each t object in $R^{\mathcal{I}}$ is connected to the same d_i object as the tuple identifiers in $\mathcal{I}'$, it holds that for all roles $R \in roles(\mathcal{T})$ and for all $(a, b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} : (a, b) \in R^{\mathcal{I}}$ if and only if $(h\,(a)\,, (b)) \in R^{\mathcal{I}'}$. Additionally, since every $t \in R^{\mathcal{I}}$ has a corresponding tuple identifier ${t_{\langle d_1,\ldots,d_n \rangle}}^{\mathcal{I}'} \in R^{\mathcal{I}'}$ and all other domain elements are mapped to the same elements, also for all concepts $C \in concepts(\mathcal{T})$ and for all $a \in \Delta^{\mathcal{I}} : a \in C^{\mathcal{I}}$ if and only if $h\,(a) \in C^{\mathcal{I}'}$ holds. Therefore, $\mathcal{I} \cong i2m_{\mathcal{T}_{R[U]}}\left(m2i_{R[U]}\,(\mathcal{I})\right)$. □

Example 4.10. We use the relational schema $course$ of Example 2.2. Let $\mathcal{T}_{course}$ be the KB given in Example 4.8. The model $\mathcal{M}$ in Figure 4.6 is a valid model of the KB $\mathcal{T}_{course}$. The function $m2i_{\mathcal{T}_{course}}\,(\mathcal{M})$ returns the instance given in Figure 2.2. The application of the function $i2m$ to that instance returns the model given in Figure 4.5 (see Example 4.9). Let us denote this model with $\mathcal{M}'$. We will now show that these two models are isomorphic. We map the domain elements of $\mathcal{M}$ and $\mathcal{M}'$ as follows:

- First, we map all domain elements not in $course^{\mathcal{M}}$, $course\_lecture^{\mathcal{M}}$, $course\_type^{\mathcal{M}}$ and $course\_room^{\mathcal{M}}$ to themselves:

$$h(\texttt{AlgebraI}^{\mathcal{M}}) = \texttt{AlgebraI}^{\mathcal{M}'} \qquad h(\texttt{Economics}^{\mathcal{M}}) = \texttt{Economics}^{\mathcal{M}'}$$
$$h(\texttt{VO}^{\mathcal{M}}) = \texttt{VO}^{\mathcal{M}'} \qquad h(\texttt{UE}^{\mathcal{M}}) = \texttt{UE}^{\mathcal{M}'}$$
$$h(\texttt{HS1}^{\mathcal{M}}) = \texttt{HS1}^{\mathcal{M}'} \qquad h(\texttt{SEM1}^{\mathcal{M}}) = \texttt{SEM1}^{\mathcal{M}'}$$

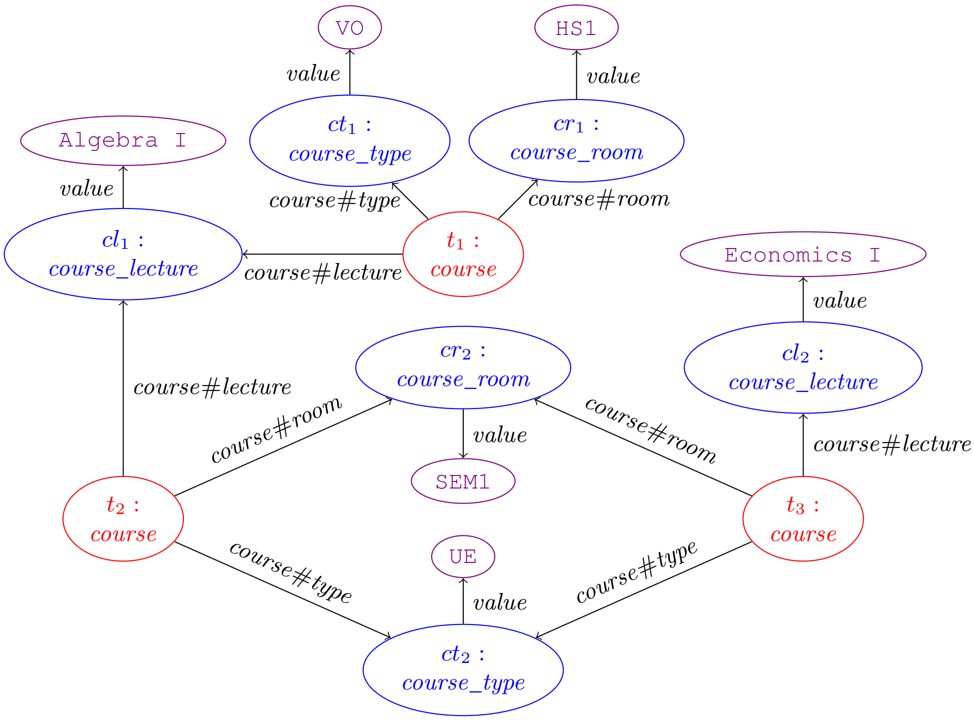


Figure 4.6: The model $\mathcal{M}_{course}$.

- Then, we map all tuple identifiers:

$$h\left(t_1^{\mathcal{M}}\right) = t^{\mathcal{M}'}_{\langle \textit{Algebra I, VO, HS1} \rangle}$$
$$h\left(t_2^{\mathcal{M}}\right) = t^{\mathcal{M}'}_{\langle \textit{Algebra I, UE, SEM1} \rangle}$$
$$h\left(t_3^{\mathcal{M}}\right) = t^{\mathcal{M}'}_{\langle \textit{Economics I, UE, SEM1} \rangle},$$

- and the value identifiers in $course\_lecture^{\mathcal{M}}$, $course\_type^{\mathcal{M}}$ and $course\_room^{\mathcal{M}}$:

$$h\left(cl_1^{\mathcal{M}}\right) = c^{\mathcal{M}'}_{\textit{lecture,Algebra I}} \qquad h\left(ct_1^{\mathcal{M}}\right) = c^{\mathcal{M}'}_{\textit{type,VO}} \qquad h\left(cr_1^{\mathcal{M}}\right) = c^{\mathcal{M}'}_{\textit{room,HS1}}$$
$$h\left(cl_2^{\mathcal{M}}\right) = c^{\mathcal{M}'}_{\textit{lecture,Economics I}} \qquad h\left(ct_2^{\mathcal{M}}\right) = c^{\mathcal{M}'}_{\textit{type,UE}} \qquad h\left(cr_2^{\mathcal{M}}\right) = c^{\mathcal{M}'}_{\textit{room,SEM1}}$$

It is easy to verify that $h\left(\cdot\right)$ is indeed an isomorphism. $\triangleleft$

We can now associate the set of instances to the set of valid models of a KB created by the function sm. As a result the next corollary directly follows from Lemma 4.6 and 4.7.

Corollary 4.1. *Let $R[U]$ be a relational schema and let $\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle$ be the result of $sm\,(R[U])$. Then*

$$Inst\,(R[U]) = \{m2i_{\mathcal{T}_{R[U]}}\,(\mathcal{I}) \mid \mathcal{I} \vDash \mathcal{T}_{R[U]} \wedge \mathcal{I} \vDash \sigma_{R[U]}\}$$

and

$$Mod\,(\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle) = Closure_{\cong}\,(\{i2m_{R[U]}\,(I) \mid I \in Inst\,(R[U])\})\,.$$

4.3 Data Dependencies

In this section we will investigate data dependencies in DLs. DLs already include different types of data dependencies. For example, concept inclusions are a form of data dependencies. Even though functionality assertions exist for DLs, it is so far not possible to model FD-like constraints over DL KBs. Functional dependencies express identification properties. In particular, in conceptual modeling it is often needed to specify that an object is uniquely identified by some of its properties. For example, a person is identified by its social security number. Thus, it is important to extend DLs with such dependencies.

Calvanese et al. introduce path-based identification constraints (pIdCs) as a mechanism for functional dependencies in *DL-Lite*$_{\mathcal{A}}$ [27]. We will introduce pIdCs in Section 4.3.1. We will investigate if we can find a direct mapping of FDs to pIdCs, such that the direct-mapping is semantics preserving. Unfortunately, in Section 4.3.2 we show that we cannot extend the direct-mapping with pIdCs such that it is semantics preserving. Therefore, we will introduce in Section 4.3.3 an extension to pIdCs, called tree-based identification constraints (tIdCs), which allows us to give a semantics preserving relational to Description Logic direct-mapping.

4.3.1 Path-based identification constraints

Path-based identification constraints (pIdCs) were introduced by Calvanese et al. [27] for various DLs. We focus on pIdCs for *DL-Lite*$_{\mathcal{A}}$ KBs. A (path-based) identification constraint states that a concept can be identified by some particular properties. These properties are paths. A path π over a *DL-Lite*$_{\mathcal{A}}$ KB is given by the following expression:

$$\pi \rightarrow S \mid D? \mid \pi \circ \pi,$$

where S denotes a basic role or attribute, D denotes a basic concept or value-domain and $\pi \circ \pi$ denotes the composition of paths. $D?$ is a test relation, which represents the identity relation on instances of D. Test relations can be used to formulate paths over instances of a specific class. For example, the path $HAS\text{-}CHILD \circ Woman?$ connects someone with his/her daughters.

A path can be seen as a complex property for an object o. An object that is reachable by π from o is called a π-*filler* for o. An object o can have several or no π-fillers. The length of a path π, denoted by $length\ (\pi)$ is inductively defined:

$$length\ (\pi) = \begin{cases} 0 & \text{if } \pi = D?, \\ 1 & \text{if } \pi = S, \\ length\ (\pi_1) + length\ (\pi_2) & \text{if } \pi = \pi_1 \circ \pi_2. \end{cases}$$

We are now able to give a definition for path-based identification constraints.

Definition 4.16. (pIdC [23, 27]) A *path-based identification constraint* (pIdC) over a *DL-Lite*$_{\mathcal{A}}$ KB is an assertion of the form

$$(\text{id } C \ \ \pi_1 \dots \pi_n)$$

where C denotes a basic concept, $n \geq 1$, and $\pi_1 \dots \pi_n$ (called the *components* of the identifier) are paths over *DL-Lite*$_{\mathcal{A}}$ such that $length(\pi_i) \geq 1$ for all $i \in [1 \dots n]$. $\triangleleft$

Such a pIdC intuitively states, that if there two objects o and o', such that they share a π_i-fillers for every $i \in [1 \dots n]$, then these two objects must be the same. For example, we can say that no two men can have the same daughters with the pIdC (id $Man \ \ HAS\text{-}CHILD \circ Woman?$). We will call a pIdC *local*, if at least one path π_i has the length of one. We can now define *DL-Lite*$_{\mathcal{A}}$ with pIdCs:

Definition 4.17. (*DL-Lite*$_{\mathcal{A},id}$ KB with pIdCs [27]) A KB in *DL-Lite*$_{\mathcal{A},id}$, that is *DL-Lite*$_{\mathcal{A}}$ with pIdCs, is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{A}$ is a *DL-Lite*$_{\mathcal{A}}$ ABox, and $\mathcal{T}$ is the union of two sets $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}_{id}$, where $\mathcal{T}_{\mathcal{A}}$ is a *DL-Lite*$_{\mathcal{A}}$ TBox, and $\mathcal{T}_{id}$ is a set of pIdCs such that

- all concepts in a pIdC of $\mathcal{T}_{id}$ are basic concepts;
- for each pIdC α in $\mathcal{T}_{id}$, every role or attribute that occurs (in either direct or inverse direction) in a path of α does not appear in the right-hand side of assertions of the form $Q \sqsubseteq Q'$ or $U \sqsubseteq U'$. $\triangleleft$

Notice that the last constraint is a generalization of the constraint already imposed over functionality assertions in *DL-Lite*$_{\mathcal{A}}$ KBs.

We now need to define the semantics of pIdCs. First, we define the semantics of a path π, which is given by an extension $\pi^{\mathcal{I}}$ in an interpretation $\mathcal{I}$ as follows:

- if $\pi = S$, then $\pi^{\mathcal{I}} = S^{\mathcal{I}}$,
- if $\pi = D?$, then $\pi^{\mathcal{I}} = \{(o, o)\, | o \in D^{\mathcal{I}}\}$,
- if $\pi = \pi_1 \circ \pi_2$, then $\pi^{\mathcal{I}} = \pi_1^{\mathcal{I}} \circ \pi_2^{\mathcal{I}}$, where $\circ$ denotes the composition operator on relations.

We denote with $\pi^{\mathcal{I}}(o)$ the set of π-fillers for o in $\mathcal{I}$. A π-filler is every object that is reachable from o in $\mathcal{I}$ by means of π, i.e. $\pi^{\mathcal{I}}(o) = \{o' \mid (o, o') \in \pi^{\mathcal{I}}\}$. Then, an interpretation $\mathcal{I}$ satisfies the IdC (id $C \ \ \pi_1 \dots \pi_n$) if for all $o, o' \in C^{\mathcal{I}}$, $\pi_1^{\mathcal{I}}(o) \cap \pi_1^{\mathcal{I}}(o') \neq \emptyset \wedge \dots \wedge \pi_n^{\mathcal{I}}(o) \cap \pi_n^{\mathcal{I}}(o') \neq \emptyset$ implies $o = o'$.

4.3.1.1 KB satisfiability with pIdCs

We will investigate *DL-Lite*$_{\mathcal{A}}$ KB satisfiability with pIdCs in the presence of weakly-acyclic KBs. It has been shown that *DL-Lite*$_{\mathcal{A},id}$ KB satisfiability with arbitrary pIdCs is NLOGSPACE -hard with respect to the ABox (see Theorem 6 of [27]), whereas KB satisfiability with local pIdCs is LOGSPACE -complete with respect to the ABox. The second is proven by a perfect reformulation of a query that asks for the violation of some pIdC in a *DL-Lite*$_{\mathcal{A},id}$ KB. If such a query returns false, i.e. no pIdC is violated, then the *DL-Lite*$_{\mathcal{A},id}$ KB is satisfiable. We will show that we can also evaluate such a query over the canonical model of a satisfiable weakly-acyclic *DL-Lite*$_{\mathcal{A}}$ KB.

First, we will define a translation of a pIdC α to a CQ with an inequality $\delta(\alpha)$ that encodes the violation of α. We will use the following translation function where B is a basic concept and x is a variable [23]:

$$\gamma(B,x) = \begin{cases} A(x), & \text{if } B = A, \\ P(x, y_{new}), & \text{if } B = \exists P, \text{ where } y_{new} \text{ is a fresh variable}, \\ P(y_{new}, x), & \text{if } B = \exists P^-, \text{ where } y_{new} \text{ is a fresh variable}. \end{cases}$$

Let the IdC α be of the form (id B $\pi_1, \ldots, \pi_n$). Then, we define a CQ with an inequality

$$\delta(\alpha)(x,x') = \exists \mathbf{x}.\gamma(B,x) \wedge \gamma(B,x') \wedge x \neq x' \wedge \bigwedge_{1 \leq i \leq n} \left(\rho(\pi_i, x, x_i) \wedge \rho(\pi_i, x', x_i)\right),$$

where $\mathbf{x}$ are all variables appearing in the atoms of $\delta(\alpha)$ except for x and x'. The translation function $\rho(\pi, x, y)$ is inductively defined on the structure of the path π as follows:

(1) If $\pi = B_1? \circ \cdots B_h? \circ Q \circ B_1'? \circ \cdots \circ B_k'?$ (with $h \geq 0, k \geq 0$), then

$$\rho(\pi, x, y) = \gamma(B_1, x) \wedge \cdots \wedge \gamma(B_h, x) \wedge Q(x,y) \wedge \gamma(B_1', y) \wedge \cdots \wedge \gamma(B_k', y).$$

(2) If $\pi = \pi_1 \circ \pi_2$, where $length(\pi_1) = 1$ and $length(\pi_2) \geq 1$, then

$$\rho(\pi, x, y) = \rho(\pi_1, x, z) \wedge \rho(\pi_2, z, y),$$

where z is a fresh variable symbol not occurring elsewhere in the query.

Intuitively, the query $\delta(\alpha)(x,x')$ asks for two different individuals x, x', such that for all $i \in [1 \ldots n]$ the path π_i starting from x and x' end at the same individual x_i. If the query returns such individuals then these two witness a violation of α. Now consider a *DL-Lite*$_{\mathcal{A},id}$ KB $\mathcal{K} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a *DL-Lite*$_{\mathcal{A}}$ TBox and $\mathcal{T}_{id}$ is a set of pIdCs. Then, the boolean UCQ

$$q_{\mathcal{T}_{id}} = \bigcup_{\alpha \in \mathcal{T}_{id}} \exists x_\alpha, x_\alpha' \left\{\delta(\alpha)(x_\alpha, x_\alpha')\right\},$$

asks for a violation of some pIdC in $\mathcal{T}_{id}$. If the query $q_{\mathcal{T}_{id}}$ is true then there is one query $\delta(\alpha)(x_\alpha, x_\alpha')$ that returns two individuals x_α and x_α', that witness the violation of some pIdC α. We can now show the following:

Theorem 4.4. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable weakly-acyclic DL-Lite$_{\mathcal{A}}$ KB, let $\mathcal{T}_{id}$ be a set of pIdCs, and let $q_{\mathcal{T}_{id}}$ be a UCQ as defined above. Then the DL-Lite$_{\mathcal{A},id}$ KB $\mathcal{K}_{id} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$ is satisfiable if and only if $q_{\mathcal{T}_{id}}^{can(\mathcal{K})} = \emptyset$.*

Proof.

($\Rightarrow$) Suppose $\mathcal{K}_{id}$ is satisfiable. We need to show that then $q_{\mathcal{T}_{id}}^{can(\mathcal{K})} = \emptyset$. Assume towards a contradiction that $q_{\mathcal{T}_{id}}^{can(\mathcal{K})}$ returns true. But then, there exists a pIdC α such that there are two constants x, x' that witness the violation of α in $can\,(\mathcal{K})$. Therefore, $\mathcal{K}_{id}$ is unsatisfiable, which contradicts our assumption.

($\Leftarrow$) Suppose $q_{\mathcal{T}_{id}}^{can(\mathcal{K})} = \emptyset$. We need to show that then $\mathcal{K}_{id}$ is satisfiable. Since no pIdC violates $can\,(\mathcal{K})$, $can\,(\mathcal{K})$ is also a model of $\mathcal{K}_{id}$. $\square$

4.3.1.2 Implication of pIdCs

We will now look at the implication problem for pIdCs. That is, given a weakly-acyclic *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}$, a set of pIdCs $\mathcal{T}_{id}$ and a pIdC α, check whether $\{\mathcal{T} \cup \mathcal{T}_{id}\} \models \alpha$, i.e. every model of $\mathcal{T}$ and $\mathcal{T}_{id}$ also satisfies α.

Additionally, we define the notion of *trivially* implied pIdCs. An FD $X \rightarrow Y$ is trivially implied if $Y \subseteq X$. Intuitively, this means that $X \rightarrow Y$ is trivially holds if X uniquely determines a subset of attributes of itself. We can generalize this notion for pIdCs. Consider the pIdC $\varphi = (\text{id } C \;\; \pi)$ where π is an arbitrary path. Then, the pIdC $(\text{id } C \;\; \pi \circ \pi^{-} \circ C?)$ is *trivially* implied by φ. This can be generalized to the following, if both $(\text{id } C \;\; \pi)$ and $(\text{id } B \;\; \pi_1 \circ C? \circ \pi)$, where π and π_1 are arbitrary paths, are in the set of implied pIdCs, then the pIdC $(\text{id } B \;\; \pi_1 \circ C? \circ \pi \circ \pi^{-} \circ C?)$ is *trivially* implied. Trivial implication in pIdCs follows from the fact, that if a concept C is uniquely determined by a path π, we can walk back the path π to the concept C. We will end at the same individual. Hence, at the concept we started from. Since, C trivially implies itself, the above holds.

We can decide implication of pIdCs by, first, creating a *DL-Lite*$_{\mathcal{A},id}$ ABox $\mathcal{A}_{\alpha}$ that violates the pIdC α. Then, we chase the *DL-Lite*$_{\mathcal{A}}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_{\alpha} \rangle$, i.e. we add all membership assertions implied by $\mathcal{T}$. Finally, we use the pIdCs in $\mathcal{T}_{id}$ and the functionality assertions in $\mathcal{T}$, denoted by $\mathcal{T}_f$, to merge constants in $chase\,(\mathcal{K})$, in order to verify the violation of α. This is implemented by the algorithm IdCImpl, illustrated in Algorithm 4.2.

input : *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$, a set of pIdCs $\mathcal{T}_{id}$, a pIdC α
output: true if $\mathcal{T} \cup \mathcal{T}_{id} \models \alpha$

$\mathcal{T}_f^{id} \rightarrow \emptyset$;
create the counter-model ABox $\mathcal{A}_\alpha$ from α;
$\mathcal{A} \leftarrow chase\,(\langle \mathcal{T}, \mathcal{A}_\alpha \rangle)$;
foreach *functionality assertion* $($*funct* $Q)$ **do**
  $\mathcal{T}_f^{id} \leftarrow \mathcal{T}_f^{id} \cup \{(\text{id } \top \; Q^-)\}$;
end
repeat
  $\mathcal{A}' \leftarrow \mathcal{A}$;
  foreach *pIdC* $\beta \in \mathcal{T}_{id} \cup \mathcal{T}_f^{id}$ **do**
    if $(x, x') \in \delta\,(\beta)^\mathcal{A}$ **then**
      if $x = x_0 \wedge x' = y_0$ **then**
        return *true*;
      else
        $\mathcal{A} \leftarrow \mathcal{A}[x/z][x'/z]$, where z is a fresh object not yet in $\mathcal{A}$;
      end
    end
  end
until $\mathcal{A}' = \mathcal{A}$;
return *false*

Algorithm 4.2: The algorithm $\mathsf{IdCImpl}$ for deciding the implication of pIdCs

For the illustration of Algorithm 4.2 we will use the following running example.

Example 4.11. Let us consider the *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$ over the concept A, and the roles P, F, S. The TBox $\mathcal{T}$ contains the concept inclusion

$$A \sqsubseteq \exists P.$$

Let the set of pIdCs $\mathcal{T}_{id}$ be the following:

$$\left(\text{id } \exists P^- \;\; P^- \circ F \circ P\right), \qquad\qquad \left(\text{id } \exists P^- \;\; P^- \circ S \circ P\right).$$

We want to check if the above implies the pIdC

$$\alpha = \left(\text{id } A \;\; P^- \circ A? \circ F \circ A? \circ S \circ A?\right),$$

i.e. $\mathcal{T} \cup \mathcal{T}_{id} \models \alpha$. $\triangleleft$

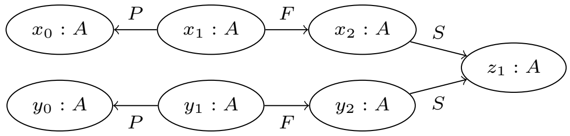


Figure 4.7: The counter-model ABox $\mathcal{A}_\alpha$.

As a first step in Algorithm 4.2 we define an ABox $\mathcal{A}_\alpha$, which we will call counter-model ABox. Given a pIdC $\alpha = (\text{id}\ \ B\ \ \pi_1, \ldots, \pi_n)$, such an ABox $\mathcal{A}_\alpha$ consists of the following set of membership assertions:

$$A_\alpha = \{\gamma(B, x_0), \gamma(B, y_0)\} \cup \bigcup_{1 \leq i \leq n} \{\varrho\left(\pi_i\left(x_0, z_i\right)\right) \cup \varrho\left(\pi_i\left(y_0, z_i\right)\right)\},$$

where $\gamma(B, x)$ is the translation function as defined in Section 4.3.1.1 and the translation function ϱ is defined on the structure of the path π as follows:

(1) If $\pi = B_1? \circ \cdots B_h? \circ Q \circ B'_1? \circ \cdots \circ B'_k?$ (with $h \geq 0, k \geq 0$), then

$$\varrho\left(\pi\left(x, y\right)\right) = \gamma\left(B_1, x\right) \cup \cdots \cup \gamma\left(B_h, x\right) \cup Q\left(x, y\right) \cup \gamma\left(B'_1, y\right) \cup \cdots \cup \gamma\left(B'_k, y\right).$$

(2) If $\pi = \pi_1 \circ \pi_2$, where $\mathit{length}\left(\pi_1\right) = 1$ and $\mathit{length}\left(\pi_2\right) \geq 1$, then

$$\varrho\left(\pi\left(x, y\right)\right) = \varrho\left(\pi_1\left(x, z\right)\right) \cup \varrho\left(\pi_2\left(z, y\right)\right),$$

where z is a fresh variable symbol not occurring elsewhere in the ABox.

Next, we use the chase to materialize the missing membership assertions, i.e. $\mathit{chase}\left(\mathcal{K}\right)$ of the KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_\alpha \rangle$.

Example 4.12. This example continues Example 4.11. The ABox $\mathcal{A}_\alpha$ built from the pIdC α is depicted in Figure 4.7. Then we chase the KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_\alpha \rangle$, which adds the membership assertions as shown in Figure 4.8. $\triangleleft$

We will now chase the membership assertions in $\mathit{chase}\left(\mathcal{K}\right)$ with the pIdCs in $\mathcal{T}_{id}$ and the functionality assertions in $\mathcal{T}$ as follows:

(1) First, we create a pIdC for each functionality assertion in $\mathcal{T}_f$ as follows:

$$\text{if } (\text{funct}\ \ Q) \in \mathcal{T}_f \text{ then } \left(\text{id}\ \ \top\ \ Q^-\right).$$

We will denote by $\mathcal{T}_f^{id}$ this newly created set of pIdCs.

(2) Then, we translate each pIdC β in $\mathcal{T}_{id} \cup \mathcal{T}_f^{id}$ to the CQ $\delta(\beta)$, which asks for a violation of β.

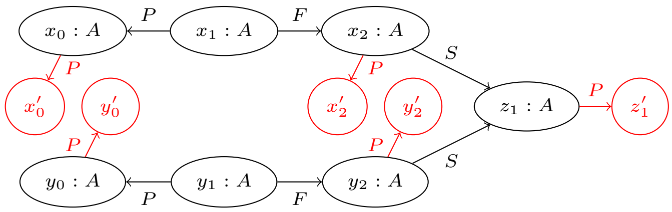


Figure 4.8: The chase of the KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_\alpha \rangle$ adds the membership assertions drawn in red.

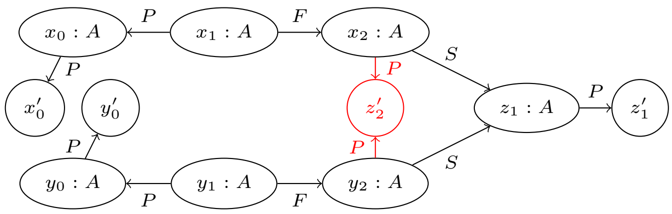


Figure 4.9: The ABox after we have applied the pIdC (id $\exists P^-$ $P^- \circ S \circ P$).

(3) If the query $\delta(\beta)$ evaluated over $DB\,(chase\,(\mathcal{A}_\alpha))$ returns a tuple of objects (x, y) that witnesses a violation of a pIdC, we substitute the objects x and y in $chase\,(\mathcal{A}_\alpha)$ with a new object z not yet in $\mathcal{A}_\alpha$.

(4) We repeat (3) until

   (a) either the objects x_0 and y_0 are returned by a CQ $\delta(\beta)$, which means that the pIdC α is implied by $\mathcal{T} \cup \mathcal{T}_{id}$ or

   (b) no further CQ returns a tuple, which means that α is not implied by $\mathcal{T} \cup \mathcal{T}_{id}$.

Example 4.13. We continue Example 4.12. We have translated each pIdC in $\mathcal{T}_{id} \cup \mathcal{T}_f^{id}$ to a CQ. The CQ of the pIdC (id $\exists P^-$ $P^- \circ S \circ P$) evaluated over the ABox in Figure 4.8 returns the tuples $\{(x'_2, y'_2), (y'_2, x'_2)\}$. Therefore, the objects x'_2 and y'_2 are removed and substituted with a new object z'_2. The resulting ABox is illustrated in Figure 4.9. This new ABox violates the pIdC (id $\exists P^-$ $P^- \circ F \circ P$). The corresponding CQ returns the tuples $\{(x'_0, y'_0), (y'_0, x'_0)\}$. Therefore, the algorithm IdCImpl returns true. Hence, $\mathcal{T} \cup \mathcal{T}_{id} \models \alpha$. $\lhd$

We will now show the correctness of Algorithm 4.2.

Theorem 4.5. *Let $\mathcal{T}$ be a weakly-acyclic satisfiable DL-Lite$_\mathcal{A}$ TBox, let $\mathcal{T}_{id}$ be a set of pIdCs and let α be a pIdC. Then, IdCImpl returns true if and only if $\mathcal{T} \cup \mathcal{T}_{id} \models \alpha$.*

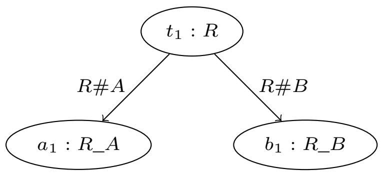
Proof idea. Notice that the algorithm IdCImpl mimics the chase with weakly-acyclic tuple-generating dependencies (tgds) and equality-generating dependencies (egds) [1,41], i.e. we can view the counter-model ABox as a database, positive inclusion dependencies in the TBox as tgds and identification constraints as egds. Then, IdCImpl and the chase coincides, therefore

	A	B
t_1	a_1	b_1

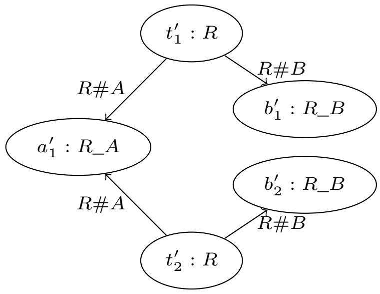
(a) Instance I_1: $I_1 \models A \rightarrow B$

	A	B
t_1'	a_1'	b_1'
t_2'	a_1'	b_2'

(b) Instance I_2: $I_2 \not\models A \rightarrow B$



(c) Model $\mathcal{M}_1 = \mathit{i2m}_{R[U]}(I_1)$: $\mathcal{M}_1 \models (\text{id } R\_B \;\; R\#B^- \circ R\#A)$



(d) Model $\mathcal{M}_2 = \mathit{i2m}_{R[U]}(I_2)$: $\mathcal{M}_2 \not\models (\text{id } R\_B \;\; R\#B^- \circ R\#A)$

Figure 4.10: Instances and their mapping into *DL-Lite*$_{\mathcal{A}}$ interpretations.

soundness and completeness of the algorithm IdCImpl can be proved similarly as soundness and completeness of the chase with weakly-acyclic tgds and egds [1, 41]. □

4.3.1.3 Path-based IdCs as a formalism to model functional dependencies in *DL-Lite*$_{\mathcal{A}}$

We will show how to model unary functional dependencies with path-based identification constraints in *DL-Lite*$_{\mathcal{A}}$. Let $R[A, B]$ be a relational schema, and let $A \rightarrow B$ be an FD over $R[A, B]$. We can map the relational schema to a *DL-Lite*$_{\mathcal{A}}$ KB using the schema to DL direct mapping introduced in Definition 4.13. We can extend this *DL-Lite*$_{\mathcal{A}}$ KB with pIdCs to express the FD $A \rightarrow B$, which is translated to the pIdC:

$$\left(\text{id } R\_B \;\; R\#B^- \circ R\#A\right).$$

Let us now consider two different instances of the relational schema $R[A, B]$. Instance I_1 given in Figure 4.10a satisfies the FD $A \rightarrow B$, where the instance I_2 in Figure 4.10b does not satisfy this FD. If we now look at the translated models given in Figures 4.10c and 4.10d, we observe the same. That is, the model $\mathcal{M}_1$ satisfies the translated pIdC and the model $\mathcal{M}_2$ does not. Unfortunately, a natural generalization for such a translation to n-ary FDs fails. This will be investigated in the upcoming section.

4.3.2 FDs and pIdCs are semantically different

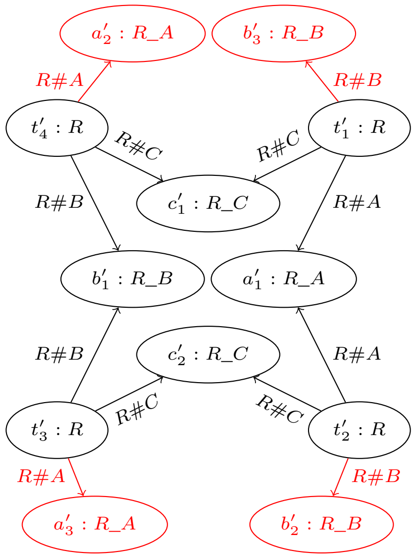
We have seen how to model unary FDs in *DL-Lite*$_{\mathcal{A}}$ KBs with pIdCs. However, this can not be generalized to non-unary FDs. We will show that two instances of a relational schema can be distinguished by FDs, i.e. one instance satisfies the FD and the other does not, but the translated pIdCs can not distinguish the translated models. The next example illustrates such a case.

	A	B	C
t'_1	a'_1	b'_2	c'_1
t'_2	a'_2	b'_1	c'_1
t'_3	a'_3	b'_1	c'_2
t'_4	a'_1	b'_3	c'_2

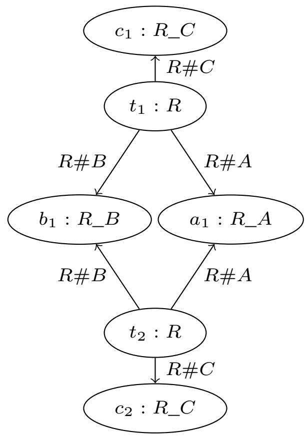
(a) Instance I_1: $I_1 \vDash \sigma$

	A	B	C
t_1	a_1	b_1	c_1
t_2	a_1	b_1	c_2

(b) Instance I_2: $I_2 \nvDash \sigma$



(c) Model $\mathcal{M}_1 = i2m_{R[U]}(I_1)$: $\mathcal{M}_1 \nvDash \delta$



(d) Model $\mathcal{M}_2 = i2m_{R[U]}(I_2)$: $\mathcal{M}_2 \nvDash \delta$

Figure 4.11: Instances and their mapping into RDF graphs. The FD $AB \rightarrow C$ can distinguish the two structures I_1 and I_2, whereas the pIdC $(\text{id}\ \ R\_C\ \ R\#C^- \circ R\#A, R\#C^- \circ R\#B)$ cannot distinguish the two models $\mathcal{M}_1$ and $\mathcal{M}_2$.

Example 4.14. Consider the FD $\sigma := AB \rightarrow C$ and the pIdC $\delta := (\text{id}\ \ R\_C\ \ R\#C^- \circ R\#A, R\#C^- \circ R\#B)$ translated from σ. We will now show that σ and δ distinguish different relational instances and *DL-Lite*$_{\mathcal{A}}$ models. In Figure 4.11 we give two instances of a relational schema $R[A, B, C]$. The instance I_1 is a valid instance satisfying the FD σ, whereas the instance I_2 is not a valid instance for the FD σ. If we now translate σ into a pIdC δ and also use $i2m$ to map the instances I_1 and I_2 to the models $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively, we get two models that both violate the translated pIdC δ (see Figure 4.11c and 4.11d). Notice that the model $\mathcal{M}_1$, without the objects and roles given in red, viewed as an ABox, is the counter-model ABox for the implication of the pIdC δ. $\lhd$

Example 4.14 just shows that δ is not a correct translation of the FD σ, s.t. $\mathcal{M}_1 \vDash \delta$ and $\mathcal{M}_2 \nvDash \delta$. In order to show that pIdCs are indeed not able to capture the differences in $\mathcal{M}_1$ and $\mathcal{M}_2$, we need to prove that for any set of IdCs Σ it holds that whenever $\mathcal{M}_1 \vDash \Sigma$, then $\mathcal{M}_2 \vDash \Sigma$. Such a proof is established in Theorem 4.6.

Theorem 4.6. *There is a set FD of functional dependencies over a relational schema $R[U]$, and a pair of relational instances I_1 and I_2 of $R[U]$, s.t. for any set Σ of pIdCs the following holds:*

(a) $I_1 \models FD$ *and* $I_2 \not\models FD$*, and*

(b) $i2m_{R[U]}(I_1) \not\models \Sigma$ *or* $i2m_{R[U]}(I_2) \models \Sigma$.

Before we proof Theorem 4.6 we need some preliminary notions. Let us first consider the following claim, which establishes the relationship between objects that are in bisimulation and their π-fillers.

Claim 4.1. *Let π be an arbitrary path in DL-Lite$_{\mathcal{A}}$, let $o_1 \in \Delta^{\mathcal{I}}$ and $o_2 \in \Delta^{\mathcal{J}}$. If $o_1 \sim_{\mathcal{B}} o_2$ then $\pi^{\mathcal{I}}(o_1) \sim_{\mathcal{B}} \pi^{\mathcal{J}}(o_2)$.*

The proof of Claim 4.1 directly follows from Definition 4.12 (Bisimulations). In order to prove Theorem 4.6, we need to have established a bisimulation between the two models in Example 4.14.

Claim 4.2. *The models $\mathcal{M}_1$ and $\mathcal{M}_2$ of Example 4.14 are in bisimulation to each other, i.e. $\mathcal{M}_1 \sim_{\mathcal{B}} \mathcal{M}_2$.*

Proof. Table 4.2 shows the domain elements that bisimulate each other in the corresponding structures. It is easily verified that the relation in Table 4.2 is indeed a bisimulation of $\mathcal{M}_1$ and $\mathcal{M}_2$. □

$\mathcal{M}_2$ \ $\mathcal{M}_1$	t'_1	t'_2	t'_3	t'_4	a'_1	a'_2	a'_3	b'_1	b'_2	b'_3	c'_1	c'_2
t_1	$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$								
t_2	$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$								
a_1					$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$					
b_1								$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$		
c_1											$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$
c_2											$\sim_{\mathcal{B}}$	$\sim_{\mathcal{B}}$

Table 4.2: Bisimulation relation of $\mathcal{M}_1 \sim_{\mathcal{B}} \mathcal{M}_2$

With a proof for the bisimulation of $\mathcal{M}_1$ and $\mathcal{M}_2$ we are ready to prove Theorem 4.6.

Proof. (of Theorem 4.6) Let $AB \to C$ be the only functional dependency in the set FD. Suppose Σ is an arbitrary set of pIdCs. Let I_1 and I_2 be the two instances of Example 4.14. We now show based on the pIdCs contained in Σ that (a) and (b) hold.

- Suppose $\Sigma = \emptyset$:

  (a) As illustrated in Example 4.14, $I_1 \models FD$ and $I_2 \not\models FD$.

(b) Clearly, $i2m_{R[U]}(I_2) \models \Sigma$ holds.

- Suppose there is an arbitrary pIdC $\sigma_C \in \Sigma$, which is of the form $(\mathrm{id}\ \ R\_C\ \ \pi_1, \ldots, \pi_n)$ in Σ:

  (a) As illustrated in Example 4.14, $I_1 \models FD$ and $I_2 \not\models FD$.

  (b) Let $\mathcal{M}_1$ denote $i2m_{R[U]}(I_1)$ and let $\mathcal{M}_2$ denote $i2m_{R[U]}(I_2)$. Observe that by Claim 4.2 $\mathcal{M}_1 \sim_{\mathcal{B}} \mathcal{M}_2$. Suppose $i2m_{R[U]}(I_2) \not\models \sigma_C$, i.e. $\mathcal{M}_2 \not\models \sigma_C$. We need to show that then $i2m_{R[U]}(I_1) \not\models \sigma_C$, i.e. $\mathcal{M}_1 \not\models \sigma_C$. Since $\mathcal{M}_2 \not\models \sigma_C$ there are two C-objects (c'_1 & c'_2), s.t. $\pi_1{}^{\mathcal{M}_2}(c'_1) \cap \pi_1{}^{\mathcal{M}_2}(c'_2) \neq \emptyset \wedge \ldots \wedge \pi_n{}^{\mathcal{M}_2}(c'_1) \cap \pi_n{}^{\mathcal{M}_2}(c'_2) \neq \emptyset$. Since $c'_1 \sim_{\mathcal{B}} c_1$ and $c'_2 \sim_{\mathcal{B}} c_2$, we will show that c'_1 and c'_2 also violate σ_C.

  In addition to Claim 4.2, we observe the following property in $\mathcal{M}_1$ and $\mathcal{M}_2$. For any object o in $\mathcal{M}_2$ and any path π, if $o' \sim_{\mathcal{B}} o$ and $o \in \pi^{\mathcal{M}_2}(c_1)$ and $o \in \pi^{\mathcal{M}_2}(c_2)$, then $o' \in \pi^{\mathcal{M}_2}(c'_1)$ and $o' \in \pi^{\mathcal{M}_2}(c'_2)$.

  Let us now denote, for an arbitrary $j \in [1 \ldots n]$, with $x \in \Delta^{\mathcal{M}_2}$ an arbitrary object in $\pi_j{}^{\mathcal{M}_2}(c'_1) \cap \pi_j{}^{\mathcal{M}_2}(c'_2)$. Since for every object $o \in \Delta^{\mathcal{M}_2}$, there is also an object $o' \in \Delta^{\mathcal{M}_1}$, such that $o \sim_{\mathcal{B}} o'$, there is also an object $y \in \Delta^{\mathcal{M}_1}$ such that $x \sim_B y$. Since $x \in \pi^{\mathcal{M}_2}(c_1)$ and $x \in \pi^{\mathcal{M}_2}(c_2)$ and the previous observation, we can conclude that $y \in \pi^{\mathcal{M}_2}(c_1)$ and $y \in \pi^{\mathcal{M}_2}(c_2)$. Therefore, $y \in \pi_j{}^{\mathcal{M}_1}(c'_1) \cap \pi_j{}^{\mathcal{M}_1}(c'_2)$. Hence, $\pi_j{}^{\mathcal{M}_1}(c'_1) \cap \pi_j{}^{\mathcal{M}_1}(c'_2) \neq \emptyset$ for all $j \in [1 \ldots n]$, which proves that $\mathcal{M}_1 \not\models \sigma_C$.

- Suppose there is an arbitrary IdC $\sigma_{AB} \in \Sigma$, which is of the form $\left(\mathrm{id}\ \ X\ \ \pi^i_1, \ldots, \pi^i_n\right)$ in Σ, where X is either R_A or R_B:

  (a) As illustrated in Example 4.14, $I_1 \models \sigma$ and $I_2 \not\models \sigma$.

  (b) Let $\mathcal{M}_2$ denote $i2m_{R[U]}(I_2)$. $\mathcal{M}_2$ has only one instance of an R_A (R_B) concept, therefore $\mathcal{M}_2 \models \sigma_{AB}$ trivially holds. □

We have now shown in Theorem 4.6, that it is possible to have two instances of a relational schema, s.t. in one instance an FD is satisfied and in the other the FD is violated. If we now translate these two instances by the schema direct mapping into models of *DL-Lite*$_{\mathcal{A}}$, we cannot find an IdC such that this IdC is satisfied in one model and violated in the other. Thus the following corollary follows from Theorem 4.6.

Corollary 4.2. *The schema direct mapping to DL-Lite*$_{\mathcal{A}}$ *KB* (sm) *extended with a mapping from FDs to pIdCs is not semantics preserving.*

The problem in the translation of the FD $AB \rightarrow C$ in Example 4.14 comes from the fact that the attributes A, B and C only refer to the columns in exactly one row. The pIdC allows one to talk about different rows. Consider the pIdC $(\mathrm{id}\ \ R\_C\ \ R\#C^- \circ R\#A, R\#C^- \circ R\#B)$. The object reachable by $R\#C^-$ in the path $R\#C^- \circ R\#A$ and in the path $R\#C^- \circ R\#B$ might be different, as illustrated in Figure 4.11c. In order to achieve a semantics preserving mapping we need to ensure that this object is the same in all paths. In the next section we propose a syntactic and semantic extension of pIdC which allows for such expressions. This extension is called tree-based identification constraints.

4.3.3 Tree-based identification constraints

In order to correctly capture the semantics of FDs, we extend path-based identification constraints to tree-based identification constraints. Let τ denote a *tree* built by the following syntax, where S denotes a role and D denotes a concept, and π denotes a path as defined in Section 4.3.1:

$$\tau \rightarrow \pi \mid \pi \circ (\tau, \ldots, \tau)$$

A tree τ evaluates over instances of concepts as follows: Let o be an object in an interpretation $\mathcal{I}$. The tuple representing the objects at the leaves of a tree τ starting from o in $\mathcal{I}$, is called a *τ-filler* for o. If the tree τ has just one leaf, i.e. it is a path, then the τ-filler coincides with the definition of a π-filler given in [27]. For convenience if $(\tau, \ldots, \tau)$ has just one path π we do not write the brackets, i.e. instead of $\pi \circ (\tau)$ we write $\pi \circ \tau$. Analogously to paths, we define the *depth* and the *width* of a tree. The depth is the equivalent to the length of paths, and inductively defined as follows:

$$depth\,(\tau) = \begin{cases} length\,(\pi) & \text{if } \tau \text{ is a path } \pi \\ length\,(\pi) + max\,(depth\,(\tau_1)\,, \ldots, depth\,(\tau_n)) & \text{if } \tau \text{ is a tree } \pi \circ (\tau_1, \ldots, \tau_n) \end{cases}$$

The width of a tree is the number of leaves and is inductively defined as follows:

$$width\,(\tau) = \begin{cases} 1 & \text{if } \tau \text{ is a path } \pi \\ \sum_{i=1}^{n} width\,(\tau_i) & \text{if } \tau \text{ is a tree } \pi \circ (\tau_1, \ldots, \tau_n) \end{cases}$$

Tree-based identification constraints (tIdCs) are an extension to pIdCs and are defined as follows.

Definition 4.18. (Tree-based identification constraints (tIdC)) A *tree-based identification constraint* over a *DL-Lite$_{\mathcal{A}}$* KB is an assertion of the form

$$(\text{id } C \;\; \tau_1, \ldots, \tau_n)$$

where C is a basic concept in *DL-Lite$_{\mathcal{A}}$*, $n \geq 1$, and $\tau_1, \ldots, \tau_n$ (called the *components* of the identifier) are trees over a *DL-Lite$_{\mathcal{A}}$* KB such that $depth\,(\tau_i) \geq 1$ for all $i \in [1 \ldots n]$. $\triangleleft$

We adapt the definition of a *DL-Lite$_{\mathcal{A},id}$* KB to include tree-based identification constraints.

Definition 4.19. (*DL-Lite$_{\mathcal{A},tid}$* KB with tIdCs) A KB in *DL-Lite$_{\mathcal{A},tid}$*, that is *DL-Lite$_{\mathcal{A}}$* with tIdCs, is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{A}$ is a *DL-Lite$_{\mathcal{A}}$* ABox, and $\mathcal{T}$ is the union of the two sets $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}_{tid}$, where $\mathcal{T}_{\mathcal{A}}$ is a *DL-Lite$_{\mathcal{A}}$* TBox, and $\mathcal{T}_{tid}$ is a set of tIdCs such that

- all concepts identified in $\mathcal{T}_{tid}$ are basic concepts;
- all concepts appearing in the test relations in $\mathcal{T}_{tid}$ are basic concepts, or basic value-domains;

- for each tIdC α in $\mathcal{T}_{tid}$, every role or attribute that occurs (in either direct or inverse direction) in a path of α does not appear in the right-hand side of assertions of the form $Q \sqsubseteq Q'$ or $U \sqsubseteq U'$. $\triangleleft$

The semantics of a tree τ is given by an extension $\tau^{\mathcal{I}}$ in an interpretation $\mathcal{I}$ as follows:

- if $\tau = \pi$, then $\tau^{\mathcal{I}} = \pi^{\mathcal{I}}$
- if $\tau = \pi \circ (\tau_1, \ldots, \tau_n)$, then

$$\begin{aligned}\tau^{\mathcal{I}} = \big\{ \big(o, \langle o_1^1, \ldots, o_k^1, \cdots, o_1^n, \ldots, o_l^n \rangle\big) \mid & \exists o'. \left(o, o'\right) \in \pi^{\mathcal{I}} \wedge \\ & \left(o', \langle o_1^1, \ldots, o_k^1 \rangle\right) \in \tau_1^{\mathcal{I}} \wedge \\ & \quad \vdots \\ & \left(o', \langle o_1^n, \ldots, o_l^n \rangle\right) \in \tau_n^{\mathcal{I}} \big\},\end{aligned}$$

where $\pi^{\mathcal{I}}$ is the extension already defined by pIdCs. The τ-filler for an object o and a tree τ, denoted by $\tau^{\mathcal{I}}(o)$, is a set of tuples with arity $\mathit{width}(\tau)$. Intuitively, the interpretation of a tree τ maps the root node of the tree with its leaves.

An interpretation $\mathcal{I}$ satisfies the tree-based identification constraint $(\mathrm{id}\ \ C\ \ \tau_1, \ldots, \tau_n)$ if for all $o, o' \in C^{\mathcal{I}}$, $\tau_1^{\mathcal{I}}(o) \cap \tau_1^{\mathcal{I}}(o') \neq \emptyset \wedge \ldots \wedge \tau_n^{\mathcal{I}}(o) \cap \tau_n^{\mathcal{I}}(o') \neq \emptyset$ implies $o = o'$. Example 4.15 illustrates tree-based identification constraints.

Example 4.15. Let us show how to distinguish the two models given in Example 4.14 with tIdCs. The translation of the FD $AB \rightarrow C$ to tIdCs is as follows:

$$\big(\mathrm{id}\ \ R\_C\ \ R\#C^- \circ (R\#A, R\#B)\big) \tag{4.36}$$

The evaluation of above IdC over the interpretation given in Figure 4.11c is given in Table 4.3. We first write the binary tuples that are in the interpretation of the roles $R\#A$, $R\#B$ and $R\#C^-$. We then, after the vertical line, combine these to tuples of objects according to the semantics of tIdCs.

Notice that the tuples in Table 4.3 correspond to the tuples in the relational instance given in Figure 4.11a. Let us check for the violation of the tIdC given in Equation 4.36. The only two different objects of type R_C are c_1' and c_2'. The τ-filler for c_1' is the set $(\langle a_1', b_3' \rangle, \langle a_2', b_1' \rangle)$ and the τ-filler for c_2' is the set $(\langle a_1', b_2' \rangle, \langle a_3', b_1' \rangle)$. The intersection of the two sets is empty, therefore the tIdC is not violated. It is easy to see that

$$\mathcal{M}_1 \models \big(\mathrm{id}\ \ R\_C\ \ R\#C^- \circ (R\#A, R\#B)\big)\,. \qquad \triangleleft$$

In comparison to pIdCs, tIdCs allows us to specify that several paths must walk through common nodes, and split afterwards. In this sense, every pIdC can be represented a tIdC, but not vice versa. Therefore, tIdCs are more expressive than pIdCs.

$R\#C^-$	$\circ\,($	$R\#A$	,	$R\#B$	$)$
(c'_1, t'_1)		(t'_1, a'_1)		(t'_1, b'_3)	
(c'_1, t'_4)		(t'_4, a'_2)		(t'_4, b'_1)	
(c'_2, t'_2)		(t'_2, a'_1)		(t'_2, b'_2)	
(c'_2, t'_3)		(t'_3, a'_3)		(t'_2, b'_1)	
		$(c'_1, \langle a'_1, b'_3\rangle)$			
		$(c'_1, \langle a'_2, b'_1\rangle)$			
		$(c'_2, \langle a'_1, b'_2\rangle)$			
		$(c'_2, \langle a'_3, b'_1\rangle)$			

Table 4.3: Evaluation of the tIdC given in Equation 4.36 over the interpretation in Figure 4.11c

4.3.3.1 KB satisfiability with tIdCs

We will investigate *DL-Lite*$_{\mathcal{A}}$ KB satisfiability with tIdCs in the presence of weakly-acyclic KBs. We will extend the method introduced for pIdCs in Section 4.3.1.1. First, we will define a translation of a tIdC α to a CQ with an inequality $\delta^t(\alpha)$ that encodes the violation of α. For paths we will use the translation ρ defined in Section 4.3.1.1.

Let the tIdC α be of the form (id B $\tau_1, \ldots, \tau_n$). Then, we define a CQ with inequality

$$\delta^t(\alpha)(x, x') = \exists \mathbf{x}.\gamma(B, x) \wedge \gamma(B, x') \wedge x \neq x' \wedge \\ \bigwedge_{1\leq i\leq n} \rho^t\left(\tau_i, x, \left\langle x^i_1, \ldots x^i_{width(\tau_i)}\right\rangle\right) \wedge \rho^t\left(\tau_i, x', \left\langle x^i_1, \ldots, x^i_{width(\tau_i)}\right\rangle\right),$$

where $\mathbf{x}$ are all variables appearing in the atoms of $\delta^t(\alpha)$ except for x and x'. The translation function $\rho^t(\tau, x, \langle x_1, \ldots, x_k\rangle)$ is inductively defined on the structure of the tree τ as follows:

(1) If $\tau = \pi$, then

$$\rho^t(\tau, x, \langle y\rangle) = \rho(\pi, x, y)$$

(2) If $\tau = \pi \circ (\tau_1, \ldots, \tau_l)$, then

$$\begin{aligned}\rho^t(\tau, x, \langle x_1, \ldots, x_k\rangle) = &\rho(\pi, x, z) \wedge \\ &\rho^t\left(\tau_1, z, \left\langle x_1, \ldots, x_{width(\tau_1)}\right\rangle\right) \wedge \\ &\vdots \\ &\rho^t\left(\tau_l, z, \left\langle x_{1+\sum_{j=1}^{l-1} width(\tau_l)}, \ldots, x_k\right\rangle\right),\end{aligned}$$

where z is a fresh variable symbol not occurring elsewhere in the query.

The query has the following intuition. First, we ask for two different individuals x and x'. These individuals must be instances of B. Additionally, they share for every tree τ in the tIdC a tuple at the leafs of τ starting in x and x'. If $\delta^t(\alpha)(x, x')$ returns two such individuals then the tIdC α is violated. Now consider a *DL-Lite*$_{\mathcal{A},tid}$ KB $\mathcal{K} = \langle \mathcal{T} \cup \mathcal{T}_{tid}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a *DL-Lite*$_{\mathcal{A}}$ TBox and $\mathcal{T}_{tid}$ is a set of tIdCs. Then, the boolean UCQs

$$q_{\mathcal{T}_{tid}} = \bigcup_{\alpha \in \mathcal{T}_{tid}} \exists x_\alpha, x'_\alpha \left\{ \delta^t(\alpha)\left(x_\alpha, x'_\alpha\right) \right\},$$

where $\mathbf{x}$ are the variables in the UCQ $q_{\mathcal{T}_{tid}}$, asks for a violation of any tIdC in $\mathcal{T}_{tid}$. We can now show the following:

Theorem 4.7. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable weakly-acyclic DL-Lite$_{\mathcal{A}}$ KB, let $\mathcal{T}_{tid}$ be a set of tIdCs, and let $q_{\mathcal{T}_{tid}}$ be a UCQs as defined above. Then the DL-Lite$_{\mathcal{A},tid}$ KB $\mathcal{K}_{tid} = \langle \mathcal{T} \cup \mathcal{T}_{tid}, \mathcal{A} \rangle$ is satisfiable if and only if $q_{\mathcal{T}_{tid}}^{can(\mathcal{K})} = \emptyset$.*

Proof. The proof is similar to the proof of Theorem 4.4 for pIdCs. □

4.3.3.2 Implication of tIdCs

The implication problem for tIdCs can be solved using the algorithm IdCImpl, established in Section 4.3.1.2. We just need to adapt the construction of the counter-model ABox for tIdCs.

Given a weakly-acyclic *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}$, a set of tIdCs $\mathcal{T}_{tid}$ and a tIdC $\alpha = (\mathrm{id}\ \ B\ \ \tau_1, \ldots, \tau_n)$, we want to check whether $\{\mathcal{T} \cup \mathcal{T}_{tid}\} \models \alpha$. We define a counter-model ABox $\mathcal{A}_\alpha$ of α consisting of the following set of membership assertions:

$$\begin{aligned} A_\alpha = &\{\gamma(B, x_0), \gamma(B, y_0)\} \cup \\ &\bigcup_{1 \le i \le n} \left\{ \varrho^t\left(\tau_i\left(x_0, \left\langle z_1^i, \ldots, z_{width(\tau_i)}^i \right\rangle\right)\right) \cup \varrho^t\left(\tau_i\left(y_1, \left\langle z_1^i, \ldots, z_{width(\tau_i)}^i \right\rangle\right)\right) \right\}, \end{aligned}$$

where $\gamma(B, x)$ is the translation function as defined in Section 4.3.1.1 and the translation function ϱ^t is defined on the structure of the tree τ as follows:

(1) If $\tau = \pi$, then

$$\varrho^t(\pi(x, \langle y \rangle)) = \varrho(\pi(x, y))$$

(2) If $\tau = \pi \circ (\tau_1, \ldots, \tau_l)$, then

$$\begin{aligned} \varrho^t(\tau(x, \langle x_1, \ldots, x_k \rangle)) = &\varrho(x, z) \cup \\ &\varrho^t\left(\tau_1\left(z, \left\langle x_1, \ldots, x_{width(\tau_1)} \right\rangle\right)\right) \cup \\ &\vdots \\ &\varrho^t\left(\tau_l\left(z, \left\langle x_{1+\sum_{j=1}^{l-1} width(\tau_j)}, \ldots, x_k \right\rangle\right)\right), \end{aligned}$$

where z is a fresh variable symbol not occurring elsewhere in the query.

Theorem 4.8. *Let $\mathcal{T}$ be a weakly-acyclic satisfiable DL-Lite$_{\mathcal{A}}$ TBox, let $\mathcal{T}_{tid}$ be a set of tIdCs and let α be a tIdC. Then,* IdCImpl*, adapted to tIdCs, returns true if and only if $\mathcal{T} \cup \mathcal{T}_{tid} \models \alpha$.*

Proof. This proof is similar to the proof of Theorem 4.5 for pIdCs. □

4.3.3.3 The Direct-Mapping of FDs to IdCs

We will now extend the schema direct mapping with a mapping from functional dependencies to tree-based identification constraints. In this section we will then prove that this mapping is semantics preserving. First, let us translate FDs to tIdCs.

Definition 4.20. (FD-direct mapping (dm)) Let U be a set of attributes $A_1, \ldots, A_n$. Given a relational schema $R\,[U]$, and a set of functional dependencies FD over $R\,[U]$, the function $dm\,(R\,[U]\,, FD)$ outputs a set of *DL-Lite$_{\mathcal{A}}$* tree-based identification assertions Σ_{FD} as follows:

Let X be a set of attributes $A_{i_1}, \ldots, A_{i_k}$. For each FD $X \to A_i \in FD$ we add a tIdC to Σ_{FD}:

$$\big(\text{id}\;\; R\_A_i\;\; R\#A_i^- \circ R? \circ (R\#A_{j_1} \circ A_{j_1}?, \ldots, R\#A_{j_k} \circ A_{j_k}?)\big)$$

The function $dm\,(R\,[U]\,, FD)$ outputs Σ_{FD}. ◁

Example 4.16. The functional dependencies in Example 2.2 are translated with the FD-direct mapping into the following tIdCs,
i.e. $dm\,(course[lecture, type, room], \{(lecture, type \to room)\,, (room \to type)\})$ outputs:

$$\begin{aligned}\big(\text{id}\;\; &course\_room\;\; course\#room^- \circ course? \circ (\;course\#lecture \circ course\_lecture?,\\ &\qquad\qquad course\#type \circ course\_type?\;))\\ \big(\text{id}\;\; &course\_type\;\; course\#type^- \circ course? \circ (\;course\#room \circ course\_room?\;))\end{aligned}$$ ◁

We now combine the FD-direct mapping with the schema direct mapping to define a direct mapping from a relational schema to a *DL-Lite$_{\mathcal{A},tid}$* TBox.

Definition 4.21. (Relational to Description Logic direct mapping (R2DM)) Given a relational schema $R\,[A_1, \ldots, A_n]$ and a set of FDs over R, the function $rdm\,(R[U], FD)$ outputs on the schema $(R\,[U]\,, FD)$ a *DL-Lite$_{\mathcal{A},tid}$* T-Box $\mathcal{T}_{R[U]}$ with tIdCs Σ as follows:

(1) First, we call $sm\,(R\,[U])$, which outputs $\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle$.

(2) Then, we call $dm\,(R\,[U]\,, FD)$, which outputs Σ_{FD}.

The function $rdm\,(R\,[U]\,, FD)$ outputs $\langle \mathcal{T}_{R[U]}, \{\sigma_{R[U]}\} \cup \Sigma_{FD} \rangle$. ◁

Example 4.17. $rdm\,(course[lecture, type, room], \{(lecture, type \to room)\,, (room \to type)\})$ outputs all assertions specified in Example 4.8 and Example 4.16. ◁

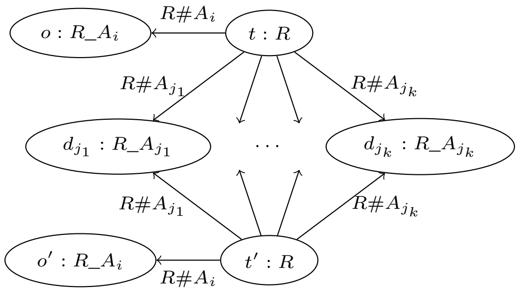


Figure 4.12: Submodel of $\mathcal{M}$, which violates φ

We have established a connection between instances of relational schemas and models of the TBox generated by the R2DM. Now, we turn our attention to the FDs. The R2DM already defines a translation of FDs to tIdCs. We want to show that our direct mapping is semantics preserving. For this, we will prove the following theorem.

Theorem 4.9. *Let I be an instance of a relational schema $R[U]$, let $\langle \mathcal{T}_{R[U]}, \sigma_{R[U]} \rangle$ be the result of $sm(R[U])$, and let FD be a set of functional dependencies over $R[U]$. Then,*

$$I \vDash FD \text{ iff } i2m_{R[U]}(I) \vDash dm(R[U], FD)$$

Proof.

(⇒) Suppose $I \vDash FD$ and assume towards a contradiction that $i2m_{R[U]}(I) \nvDash dm(R[U], FD)$. Then there exists some IdC $\varphi \in dm(R[U], FD)$, for which it holds that $i2m_{R[U]}(I) \nvDash \varphi$. By the definition of dm, φ is of the form

$$\left(\text{id } R\_A_i \;\; R\#A_i^- \circ R? \circ (R\#A_{j_1} \circ A_{j_1}?, \dots, R\#A_{j_k} \circ A_{j_k}?)\right),$$

representing the functional dependency $A_{j_1}, \dots, A_{j_k} \to A_i$. Let π denote the tree in the IdC φ. Since $i2m_{R[U]}(I) \nvDash \varphi$, the model $\mathcal{M}$ outputted by $i2m_{R[U]}(I)$ has two distinct R_A_i objects o, o', with $\pi^{\mathcal{M}}(o) \cap \pi^{\mathcal{M}}(o') \neq \emptyset$. Let $\{d_{j_1}, \dots, d_{j_k}\} \in \pi^{\mathcal{M}}(o) \cap \pi^{\mathcal{M}}(o')$. Figure 4.12 illustrates the submodel of $\mathcal{M}$, which leads to a violation of φ.

We now apply $m2i$ to $\mathcal{M}$ and by Lemma 4.6 $m2i_{\mathcal{T}_{R[U]}}(\mathcal{M}) = I$. This instance I has two tuples t and t', s.t. $t[A_i] = o$ and $t'[A_i] = o'$. Additionally $t[A_{j_i}] = t[A_{j_i}]$, for all $i \in [1 \dots k]$. Therefore, $A_{j_1}, \dots, A_{j_k} \to A_i$ is not valid in I, i.e. $I \nvDash FD$, a contradiction.

(⇐) Suppose $i2m_{R[U]}(I) \vDash dm(R[U], FD)$ and assume towards a contradiction that $I \nvDash FD$. Then, there is an FD $\sigma \in FD$, s.t. $I \nvDash \sigma$, where $\sigma = A_{j_1}, \dots, A_{j_k} \to A_i$. Therefore, I has two tuples t, t' with different values in the A_i columns, but the tuples agree on the values in the $A_{j_1}, \dots, A_{j_k}$ columns, i.e. $t[A_i] \neq t'[A_i]$ and $t[A_{j_i}] = t'[A_{j_i}]$ for all $i \in [1 \dots k]$. $i2m_{R[U]}(I)$ outputs a model $\mathcal{M}$, with two tuple identifiers t and t', s.t. t and t' are connected to the same $R\_A_{j_1}, \dots, R\_A_{j_k}$ objects, but to different R_A_i objects (compare to Figure 4.12). The function dm also translates σ into the IdC

$$\varphi_\sigma := \left(\text{id } R\_A_i \;\; R\#A_i^- \circ R? \circ (R\#A_{j_1} \circ A_{j_1}?, \dots, R\#A_{j_k} \circ A_{j_k}?)\right).$$

Since the submodel of $\mathcal{M}$ depicted in Figure 4.12 violates φ_σ, also $\mathcal{M} \not\vDash \varphi_\sigma$. This contradicts the assumption that $i2m_{R[U]}(I) \vDash dm(R[U], FD)$. Therefore, $I \vDash FD$ □

Corollary 4.3. *Let* $(R[U], FD)$ *be a relational schema, let* $\langle \mathcal{T}_{R[U]}, \Sigma \rangle$ *be the result of* $drm(R[U], FD)$ *and let* $\mathcal{M}$ *be a model of* $\langle \mathcal{T}_{R[U]}, \Sigma \rangle$. *Then,*

$$\mathcal{M} \vDash \Sigma \textit{ iff } m2i_{\mathcal{T}_{R[U]}}(\mathcal{M}) \vDash FD$$

Proof. Corollary 4.3 follows from Theorem 4.9 and Corollary 4.1 □

From Theorem 4.9 and Corollary 4.3 it follows that the R2DM is **semantics preserving**.

4.4 Normal Forms

In the previous section we have established tree-based identification constraints for modeling functional dependencies in *DL-Lite*$_{\mathcal{A}}$ knowledge bases. We will now look for a generalization of BCNF, similar to XNF, for *DL-Lite*$_{\mathcal{A},tid}$ KBs. BCNF describes redundancy based on FDs, XNF uses XFDs and we will look for redundancies based on tIdCs. In this section we will first look at what a "redundancy" is in the context of *DL-Lite*$_{\mathcal{A},tid}$ KBs. Based on those insights, we will define Description Logic Normal Form (DLNF). In Section 4.5 we will prove that whenever a relational schema is in BCNF, then the *DL-Lite*$_{\mathcal{A}}$ KB, translated from this schema, is in DLNF.

4.4.1 Redundancy in *DL-Lite*$_{\mathcal{A},tid}$ KBs

Let us first look at the redundancy in the relational instance depicted in Figure 2.2, which is not in BCNF, as it is illustrated in Example 2.2. The FD $room \rightarrow type$ violates BCNF, thus $room$ is not a superkey of the relation $course$. The translation of this instance via the R2DM is given in Figure 4.5. The translated tIdC is

$$\sigma := \left(\text{id } \; course\_type \;\; course\#type^{-} \circ course? \circ course\#room \circ course\_room?\right). \tag{4.37}$$

We can query the information expressed by this tIdC using a modified CQ, generated by the translation of tIdCs to CQ. Such a query asks for all course types and rooms in a model and looks as follows:

$$q_\sigma(t, x, r) \leftarrow course\_type(t), course\#type(t, x), course(x), \tag{4.38}$$
$$course\#room(x, r), course\_room(r) \tag{4.39}$$

The query q_σ over the model given in Figure 4.5 returns the following tuples:

t	x	r
$c_{type,VO}$	$t_{\langle Algebra\ I,\ VO,\ HS1\rangle}$	$c_{room,HS1}$
$c_{type,UE}$	$t_{\langle Algebra\ I,\ UE,\ SEM1\rangle}$	$c_{room,SEM1}$
$c_{type,UE}$	$t_{\langle Economics\ I,\ UE,\ SEM1\rangle}$	$c_{room,SEM1}$

The information that each room can only host courses of a particular type, enforced by the tIdC σ, is stored redundantly. If we now want to specify that the only lecture type in room "SEM1" is "VO", we need to update the role membership assertions of $course\#type$ several times. Thus, updating just one role membership assertion leads to an update anomaly.

How can we avoid such a redundancy? BCNF asks if the left-hand side of an FD is a superkey of the relation. XML Normal Form asks that if some attribute a is uniquely determined by another set of attributes, then the parent element of a should also be uniquely determined by the same set of attributes. In *DL-Lite*$_{\mathcal{A},tid}$ KBs we neither have a flat structure as in the relational model nor a hierarchical structure as in XML documents. The graph-like structure of *DL-Lite*$_{\mathcal{A},tid}$ allows us to talk about the neighbors of an object. If we view XML documents as graphs, the parent element of an attribute can also be considered as neighbor. Therefore, we want to define *DL-Lite*$_{\mathcal{A},tid}$ normal form based on the neighbors of an object, i.e. for each object a that is uniquely determined by a set of objects reachable via a tree its neighboring objects are also uniquely determined by the same set of objects. We will formalize this notion in the next section.

4.4.2 *DL-Lite*$_{\mathcal{A},tid}$ Normal Form

Before we define *DL-Lite*$_{\mathcal{A},tid}$ Normal Form, we need some preliminary notions. In particular, we need to define the set $neighbors$ of a tree τ and the $subtrees$ of a tree τ.

Definition 4.22. (subtrees of τ) Let τ be a tree. Then, we denote by $subtrees\,(\tau, i)$ the subtrees of τ starting at depth i, where $0 \leq i \leq depth\,(\tau) - 1$. $\triangleleft$

Definition 4.23. (neighbors of τ) Let τ be a tree. Then, we denote by $neighbors\,(\tau)$ the concepts appearing at $depth$ 1 in τ. If this is a concept test $B?$, then B is in the set $neighbors\,(\tau)$. If this is a role R then $\exists R$ is in the set $neighbors\,(\tau)$. $\triangleleft$

Let σ be a tIdC. Then, we denote by $\Pi\,(\sigma)$ the components of σ. The neighbors of a tIdC σ, denoted by $neighbors\,(\sigma)$, is the set of neighbors of all trees in $\Pi\,(\sigma)$.

Example 4.18. Let τ be the first component of the tIdC σ given in Equation 4.37. Then, $subtrees\,(\tau, 1)$ is the tree $course? \circ course\#room \circ course\_room?$. Since σ has only one component, the neighbors of σ are the same as the neighbors of τ, i.e. $neighbors\,(\tau) = \{course\}$. $\triangleleft$

We are now ready to define Description Logic Normal Form for *DL-Lite*$_{\mathcal{A},tid}$ KBs.

Definition 4.24. (Description Logic Normal Form (k-DLNF)) Let $\mathcal{T}$ be a *DL-Lite*$_{\mathcal{A}}$ TBox and let Φ be a set of tIdCs over $\mathcal{T}$. Then $\langle \mathcal{T}, \Phi \rangle$ is in k-DLNF if and only if for every nontrivial tIdC φ, s.t. $\langle \mathcal{T}, \phi \rangle \models \varphi$ and the depth of every component in φ is at most k, it is the case that for each $C \in neighbors(\varphi)$ it holds that $\langle \mathcal{T}, \Phi \rangle \models (\mathrm{id}\ \ C\ \ \Pi'(C))$, where

$$\Pi'(C) = \{subtrees\,(\tau, 1) \mid neighbors\,(\tau) = C \wedge depth\,(\tau) > 1\ \ \forall \tau \in \Pi\,(\varphi)\}\,. \quad \triangleleft$$

If k is arbitrarily large we simply say that a KB with tIdCs is in DLNF. Notice that, every TBox $\mathcal{T}$ with tIdCs Φ is in 1-DLNF. k-DLNF captures the intuition of a normal form for DLs given in the previous section. We said that if a concept C is uniquely determined by another set of concepts, the neighbors of C must be uniquely determined by the same set of concepts. Since tIdCs translated by the FD-direct mapping are of depth 2, we consider 2-DLNF as an equivalent notion for BCNF in DLs. In Section 4.5 we will show that this is indeed the case.

It is important to talk only about nontrivial tIdCs, since every functional dependency $(\text{funct}\ \ R)$ implies the tIdC $(\text{id}\ \ \top\ \ R^{-} \circ R)$. Then, it might not be the case that also $(\text{id}\ \ \exists R\ \ R)$ is implied. If we would force that $(\text{id}\ \ \exists R\ \ R)$ is implied, then it would not be possible for two different individuals to be connected with an R role to the same individual. For example, let us assume that the role $\mathit{firstname}$ connects a concept person with its first name, hence $\mathit{firstname}$ is functional. If we also include trivial tIdCs this would imply, that all people with the same first name have to be the same persons.

We will now look at several examples. The first example shows a KB translated by the R2DM from a relational schema that is not in BCNF.

Example 4.19. Let us consider the relational schema $course\,(lecture, type, room)$ as introduced in Example 2.2. This relational schema is not in BCNF. We will show that the translation of this schema is also not in 2-DLNF. The FD $room \rightarrow type$ leads to a violation of BCNF. The translated tIdC is given in Equation 4.37. We need to show that

$$\sigma' = (\text{id}\ \ course\ \ course? \circ course\#room \circ course\_room?)$$

is also implied by the TBox $\mathcal{T}_{course}$ given in Example 4.8 and the set of tIdCs Σ_{course} given in Example 4.16. We have seen in Example 4.9 that the interpretation depicted in Figure 4.5 is a model of $\langle \mathcal{T}_{course}, \Sigma_{course} \rangle$. Hence, it should also be a model of σ'. Unfortunately, this is not the case. The objects $t_{\langle \textit{Algebra I, UE, SEM1} \rangle}$ and $t_{\langle \textit{Economics I, UE, SEM1} \rangle}$ are both identified by the object $c_{\textit{room,SEM1}}$. Therefore, $course$ is not in 2-DLNF.

In the relational model a repair of the relational schema that is dependency preserving is not possible. In Example 3.14 we have seen a XML document of the same information that is both information and dependency preserving. The same information on courses was already modeled with the TBox $\mathcal{T}_c$ given in Example 4.2. The translation of the FDs $room \rightarrow type$ and $room \rightarrow building$ are already covered by the functionality assertion $(\text{funct}\ for)$ and $\big(\text{funct}\ \ has\_room^{-}\big)$, respectively. Therefore, we only need to specify a tIdC that models the FD $lecture, type \rightarrow room$, which is:

$$\big(\text{id}\ \ room\ \ located^{-}, for\big)\,. \tag{4.40}$$

We will now check if $\mathcal{T}_c$ is in 2-DLNF. Additionally, to the tIdC in Equation 4.40 the function-

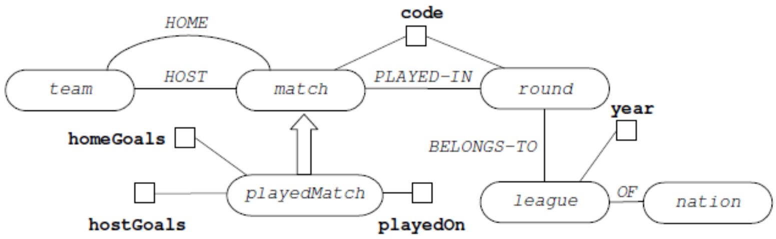


Figure 4.13: Diagrammatic representation of the football leagues KB [27].

ality assertions in $\mathcal{T}_c$ imply the following tIdCs:

$$\left(\text{id } \top \ \mathit{for}^-\right) \tag{4.41}$$
$$(\text{id } \top \ \mathit{has\_room}) \tag{4.42}$$
$$\left(\text{id } \top \ \mathit{located}^-\right) \tag{4.43}$$
$$\left(\text{id } \top \ @name^-\right) \tag{4.44}$$

Furthermore, this set of tIdCs implies the following tIdCs of depth 2:

$$\left(\text{id } \top \ \mathit{for}^- \circ \mathit{located}^-\right) \tag{4.45}$$
$$\left(\text{id } \top \ \mathit{has\_room} \circ \mathit{located}^-\right) \tag{4.46}$$
$$\left(\text{id } \mathit{room} \ \mathit{located}^-, \mathit{for} \circ \mathit{for}^-\right) \tag{4.47}$$

It is easy to see that for this set of tIdCs the condition imposed by 2-DLNF holds, i.e. the tIdCs $\left(\text{id } \exists \mathit{located}^- \ \mathit{located}^-\right)$ and $\left(\text{id } \exists \mathit{for}^- \ \mathit{for}^-\right)$ are also implied by the above set of tIdCs. Notice that the ABox $\mathcal{A}_c$ given in Figure 4.1 viewed as an interpretation is a model of $\mathcal{T}_c$ and does not contain any redundant information. ◁

The second example shows how to check DLNF for an arbitrary *DL-Lite*$_{\mathcal{A},tid}$ KB. Additionally, it recapitulates the intuition of DLNF.

Example 4.20. (Football league [27])
Consider the football leagues KB from [27] depicted in Figure 4.13. Over this KB a possible tree-based identification assertion is

$$\left(\text{id } \mathit{league} \ \textbf{year}, \mathit{BELONGS\text{-}TO}^- \circ \mathit{PLAYED\text{-}IN}^- \circ \mathit{HOME}\right),$$

which says that no home team plays in different leagues in the same year [27]. In order to test if the ontology in Figure 4.13 is in DLNF, we have to prove that

$$\left(\text{id } \mathit{round} \ \mathit{PLAYED} - \mathit{IN}^- \circ \mathit{HOME}\right)$$

is implied by the IdCs of the ontology. Such IdC states that no home team plays in different rounds, which is an implausible constraint. Therefore, the above ontology is not in DLNF. Now consider the BCNF intuition "Do Not Represent the Same Fact Twice" and the valid (up to the

l	y	t
l1	2013	t1
l1	2013	t1

Table 4.4: Answers to the CQ 4.48 over the ontology instance in Figure 4.14.

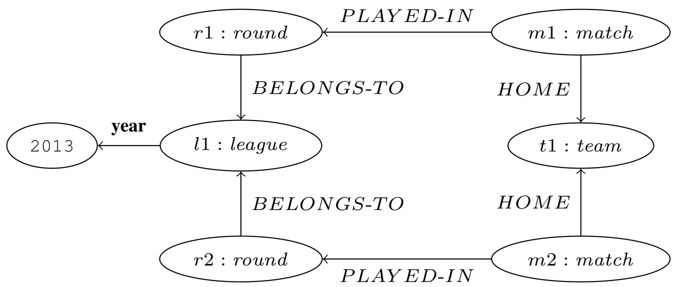


Figure 4.14: Diagrammatic representation of an ABox of the football leagues ontology.

missing concepts) instance of the ontology depicted in Figure 4.14. Now consider again the tIdC (id $league$ **year**, $BELONGS\text{-}TO^- \circ PLAYED\text{-}IN^- \circ HOME$). As we have seen, during the chase for implication of tIdCs, we can formulate a tIdC as a conjunctive query. Let us now consider the answer to the CQ 4.48 over the ABox illustrated in Figure 4.14 viewed as an interpretation. These answers are given in Table 4.4. We notice, that we have as answers two times the same information, which, having the BCNF intuition in mind, coincides to our intuition that the tIdC stated above leads to a violation of DLNF. $\triangleleft$

$$league\_id(l, y, t) \leftarrow league(l) \wedge year(l, y) \wedge BELONGS\text{-}TO^-(l, x) \wedge PLAYED\text{-}IN^-(x, y) \wedge HOME(y, t) \tag{4.48}$$

These examples give us the following intuition for DLNF. Whenever a concept is uniquely determined by another set of concepts, then these concepts have to be reachable by a unique path or tree. With this observation one can conclude that for a *DL-Lite*$_{\mathcal{A},tid}$ KB if all roles appearing in tIdCs are functional then this KB is in DLNF.

4.5 BCNF - DLNF

Finally, we want to show that our definition of DLNF corresponds to BCNF in the relational model. This means that if a relational schema is in BCNF then also the *DL-Lite*$_{\mathcal{A},tid}$ KB generated by the R2DM is in 2-DLNF and vice versa. This is captured by the following theorem.

Theorem 4.10. *Let $R[U]$ be a relational schema and FD a set of functional dependencies over $R[U]$. Let $\langle \mathcal{T}_{R[U]}, \Sigma \rangle$ denote the output of the function $drm\ (R[U], FD)$. Then $(R[U], FD)$ is in BCNF iff $\langle \mathcal{T}_{R[U]}, \Sigma \rangle$ is in 2-DLNF.*

Before we start with the proof for the theorem we observe the following. According to the relational-direct mapping we only have the two types of tIdCs in the set Σ_{FD}:

- $\big(\text{id } R\_A_i \;\; R\#A_i^- \circ (R\#A_{i_1}, \ldots, R\#A_{i_n})\big)$,
- $(\text{id } R \;\; R\#A_1, \ldots, R\#A_n)$.

Also notice that, because of $(\text{funct } R\#A_i)$ in the TBox of the R2DM the tIdCs

$$\big(\text{id } \top \;\; R\#A_i^- \circ (R\#A_{i_1}, \ldots, R\#A_{i_n})\big)$$

and

$$\big(\text{id } R\_A_i \;\; R\#A_i^- \circ R\#A_{i_1}, \ldots, R\#A_i^- \circ R\#A_{i_n}\big)$$

are equivalent. Additionally, because of concept disjointness, we never encounter in a tree an inverse role only after either the same forward role, or at the beginning of a path or tree. For example, $\big(\text{id } R \;\; R\#A_1 \circ R\#A_2^-\big)$ is satisfied in all models of the created TBox, since the object after $R\#A_1$ would be inferred to be an instance of the concept R_A_1 and R_A_2, which contradicts the TBox assertion: $R\_A_1 \sqsubseteq \neg R\_A_2$. We also need the following lemma:

Lemma 4.8.

$$A_1, \ldots, A_k \to B \in (R[U], FD)^+$$

if and only if

$$\big\langle \mathcal{T}_{R[U]}, \Sigma_{FD} \big\rangle \models \big(\textit{id } R\_B \;\; R\#B^- \circ R? \circ (R\#A_1, \ldots, R\#A_k)\big)\,.$$

Proof. Follows from Theorem 4.9 and Corollary 4.3. □

And finally, we can establish a proof for Theorem 4.10:

Proof.

($\Leftarrow$) Suppose $\langle \mathcal{T}_R, \Sigma_{FD}\rangle$ is in DLNF. We have to show that $(R[U], FD)$ is in BCNF. Suppose that there are attributes $\{A_{i_1}, \ldots, A_{i_n}, A_i\} \subseteq U$, s.t. $A_{j_1}, \ldots, A_{j_k} \to A_i$ is a nontrivial functional dependency in $(R[U], FD)^+$. We have to prove that $A_{j_1}, \ldots, A_{j_k} \to U \in (R[U], FD)^+$. By Lemma 4.8 we know that $\langle \mathcal{T}_R, \Sigma_{FD}\rangle \models \big(\text{id } R\_A_i \;\; R\#A_i^- \circ (R\#A_{j_1}, \ldots, R\#A_{j_k})\big)$. Since, $\langle \mathcal{T}_R, \Sigma_{FD}\rangle$ is in 2-DLNF and $neighbors(R\_A_i) = \{R\}$, also $\langle \mathcal{T}_R, \Sigma_{FD}\rangle \models (\text{id } R \;\; R\#A_{j_1}, \ldots, R\#A_{j_k})$. Since $(\text{funct } R\#A_i) \models \big(\text{id } \top \;\; R\#A_i^-\big)$ also $\langle \mathcal{T}_R, \Sigma_{FD}\rangle \models \big(\text{id } A_i \;\; R\#A_i^- \circ (R\#A_{j_1}, \ldots, R\#A_{j_k})\big)$ for all $A_i \in U$. By Lemma 4.8 also $A_{i_1}, \ldots, A_{i_k} \to A_i$ for all $A_i \in U$, which proves that $(R[U], FD)$ is in BCNF.

($\Rightarrow$) Suppose $(R[U], FD)$ is in BCNF. We have to show that $\langle \mathcal{T}_R, \Sigma_{FD} \rangle$ is in 2-DLNF. We distinguish two cases:

- Let $\varphi_1 = \big(\text{id}\ \ R\_A_i\ \ R\#A_i^- \circ (R\#A_{i_1}, \ldots, R\#A_{i_k})\big)$, such that $\langle \mathcal{T}_R, \Sigma_{FD} \rangle \models \varphi_1$:

  We need to show that $\langle \mathcal{T}_R, \Sigma_{FD} \rangle \models (\text{id}\ \ R\ \ R\#A_{i_1}, \ldots, R\#A_{i_k})$. By Lemma 4.8 $A_{i_1}, \ldots, A_{i_k} \rightarrow A_i \in (R[U], FD)^+$. Since $(R[U], FD)$ is in BCNF, also $A_{i_1}, \ldots, A_{i_k} \rightarrow U \in (R[U], FD)^+$, i.e. for all $A_l \in U$ $A_{i_1}, \ldots, A_{i_k} \rightarrow A_l \in (R[U], FD)^+$. Therefore, by Lemma 4.8, for all R_A_l,

  $$\langle \mathcal{T}_R, \Sigma_{FD} \rangle \models \big(\text{id}\ \ R\_A_l\ \ R\#A_l^- \circ (R\#A_{i_1}, \ldots, R\#A_{i_k})\big)\,.$$

  This IdC together with the IdC $(\text{id}\ \ R\ \ R\#A_1, \ldots, R\#A_n)$ imply that

  $$\langle \mathcal{T}_R, \Sigma_{FD} \rangle \models (\text{id}\ \ R\ \ R\#A_{i_1}, \ldots, R\#A_{i_k})\,,$$

  which proves that $\langle \mathcal{T}_R, \Sigma_{FD} \rangle$ is in 2-DLNF.

- Let $\langle \mathcal{T}_R, \Sigma_{FD} \rangle \models \Big(\text{id}\ \ R\ \ R\#A_{j_1} \circ R\#A_{j_1}^-, \ldots, R\#A_{j_k} \circ R\#A_{j_k}^-\Big)$:

  We need to show for all $i \in [1 \ldots k]$ that $\langle \mathcal{T}_R, \Sigma_{FD} \rangle \models (\text{id}\ \ R\_A_{j_i}\ \ R\#A_{j_i})$. Since $(\text{funct}\ R\#A_i)$ is in $\mathcal{T}_R$ and is equivalent to $\big(\text{id}\ \top\ R\#A_i^-\big)$, the IdCs $(\text{id}\ \ R\_A_{j_i}\ \ R\#A_{j_i})$ are trivially implied by $\mathcal{T}_R$. Therefore $\langle \mathcal{T}_R, \Sigma_{FD} \rangle$ is in 2-DLNF. $\square$

4.6 Summary

In this chapter we have recalled the Description Logic *DL-Lite*$_\mathcal{A}$ as a formalism for graph databases. A *DL-Lite*$_\mathcal{A}$ KB is constituted of a *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$, which specifies general knowledge of a domain of interest, and a *DL-Lite*$_\mathcal{A}$ ABox, which specifies knowledge of individuals in a domain. The models of a *DL-Lite*$_\mathcal{A}$ KB are given in terms of interpretations. We have considered different reasoning services in *DL-Lite*$_\mathcal{A}$, among them are KB satisfiability and query answering. For KB satisfiability we have introduced the notion of a *DL-Lite*$_\mathcal{A}$ chase. We have seen that the chase terminates if the PI in the KB are weakly-acyclic. For query answering we have given two different methods. On the one hand, the chase can be used to materialize the canonical model, which then allows one to directly query this model. On the other hand, the perfect rewriting method allows one to include all assertions of a TBox into the query, which is then evaluated over the ABox.

We have then introduced a direct-mapping from a relational schema to a *DL-Lite*$_\mathcal{A}$ TBox. Additionally, we can also translate instances of a relational schema to models of such a *DL-Lite*$_\mathcal{A}$ TBox. Since an equivalent to functional dependencies is missing in *DL-Lite*$_\mathcal{A}$, we introduced path-based identification constraints. We have investigated KB satisfiability and implication of pIdCs in *DL-Lite*$_\mathcal{A}$. Unfortunately, pIdCs are not the ideal candidate. It was shown that the direct-mapping extended with pIdCs is not semantics preserving. Therefore, we introduced tree-based identification constraints as an extension to pIdCs. KB satisfiability and implication of tIdCs can be solved similar as with pIdCs. We have then shown that the direct-mapping extended with tIdCs, called relational to Description Logic direct-mapping (R2DM) is semantics preserving.

Finally, we investigated redundancies in *DL-Lite*$_{\mathcal{A}}$ and established k-DLNF as an analogon to BCNF in *DL-Lite*$_{\mathcal{A}}$ with tIdCs. We have shown that if a relational schema is in BCNF then the *DL-Lite*$_{\mathcal{A}}$ KB, translated by the R2DM from the relational schema, is in 2-DLNF and vice versa.

CHAPTER 5

Conclusion

5.1 Discussion

In this thesis we have investigated database design in different data models: the relational model, XML documents and Description Logic Knowledge Bases. One important goal of database design is to avoid redundancies arising from badly designed models. In the relational model we have focused on FDs. Normal Forms, especially Boyce-Codd Normal Form, avoid redundancies arising from FDs. For XML documents we have summarized the work by Arenas and Libkin [8]. They have introduced XFDs and XML Normal Form.

As the data available in graph databases, especially in the Semantic Web, grows, it is needed to focus on consistency and redundancy in such data models. Therefore, data design must also play an important role in graph databases. We have used ideas from the work on normal forms in the relational model and XML documents in order to find a normal form for graph databases.

First, we have fixed $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ as a formal model for graph databases. We introduced the relational to Description Logic direct mapping for translating relational schemas to $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ KBs. We considered path-based identification constraints [27] as a formalism to model FDs in $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ KBs. We then showed that the direct-mapping extended with pIdCs is not semantics preserving. Therefore, we extended pIdCs and introduced tree-based identification constraints. We showed that the direct-mapping extended with tIdCs is indeed semantics preserving.

Tree-based identification constraints allowed us to introduce Description Logic Normal Form (k-DLNF). DLNF tries to avoid redundancies in DL KBs analogously to BCNF in relational databases. As we have shown in Section 4.4, a relational schema is in BCNF if and only if the $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ KB, translated from this relational schema, is in 2-DLNF.

5.2 Future Work

As an extension of this work in the future we will focus on at least three major topics:

- First, it is needed to thoroughly investigate further *properties of tree-based identification constraints*. We have given an algorithm to decide the implication problem for tIdCs over weakly-acyclic KBs. It remains to show, if there is an algorithm for the implication problem of tIdCs over arbitrary KBs. The same also holds for pIdCs. Another open question is, whether there exists an inference system, similar to Armstrong Axioms, for pIdCs.

- Second, we need to *extend the theory for DLNF*. We have defined DLNF as a faithful translation of BCNF to DLs. The most important question is, if there is an algorithm that efficiently checks if a *DL-Lite*$_{\mathcal{A}}$ KB is in DLNF. This involves the computation of the closure of tIdCs, which is also a problem open to be solved. Furthermore, it is interesting to look for a decomposition algorithm that repairs *DL-Lite*$_{\mathcal{A}}$ KBs which are not in DLNF.

- In this work we have extensively studied the relationship between the relational model and DL KBs. It remains to study also *the relationship between XML documents and DL KBs*. Therefore, it would be interesting to find a direct-mapping from XML documents to *DL-Lite*$_{\mathcal{A}}$ KBs. We can then ask if it also holds that whenever an XML document is in XNF, the *DL-Lite*$_{\mathcal{A}}$ KB, translated from this XML document, is in DLNF and vice versa.

Bibliography

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] R. Angles and C. Gutiérrez. Survey of graph database models. *ACM Computing Surveys*, 40(1), 2008.

[3] M. Arenas. *Design Principles for XML Data*. PhD thesis, Toronto, Canada, 2005.

[4] M. Arenas. Normalization theory for XML. *SIGMOD Record*, 35(4):57–64, 2006.

[5] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. F. Sequeda. A Direct Mapping of Relational Data to RDF. `http://www.w3.org/TR/rdb-direct-mapping`, May 2012.

[6] M. Arenas and L. Libkin. A Normal Form for XML Documents. In *PODS*, pages 85–96, 2002.

[7] M. Arenas and L. Libkin. An information-theoretic approach to normal forms for relational and XML data. In *PODS*, pages 15–26, 2003.

[8] M. Arenas and L. Libkin. A Normal Form for XML Documents. *ACM Transactions on Database Systems*, 29(1):195–232, March 2004.

[9] W. W. Armstrong. Dependency Structures of Data Base Relationships. In *IFIP Congress*, pages 580–583, 1974.

[10] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The DL-Lite Family and Relations. *J. Artif. Intell. Res. (JAIR)*, 36:1–69, 2009.

[11] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, 2000.

[12] F. Baader. Description logics. In S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M.-C. Rousset, and R. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems*, volume 5689 of *Lecture Notes in Computer Science*, pages 1–39. Springer Berlin Heidelberg, 2009.

[13] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[14] P. B. Baeza. Querying graph databases. In Hull and Fan [46], pages 175–188.

[15] P. B. Baeza, M. Romero, and M. Y. Vardi. Semantic acyclicity on graph databases. In Hull and Fan [46], pages 237–248.

[16] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. `http://www.w3.org/TR/owl-ref/`, February 2004.

[17] C. Beeri and P. A. Bernstein. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Trans. Database Syst.*, 4(1):30–59, 1979.

[18] C. Beeri, P. A. Bernstein, and N. Goodman. A sophisticate's introduction to database normalization theory. In S. B. Yao, editor, *VLDB*, pages 113–124. IEEE Computer Society, 1978.

[19] D. Berardi, D. Calvanese, and D. Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168:70–118, 2005.

[20] T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[21] P. A. Bernstein. Errata: Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Trans. Database Syst.*, 4(3):396, 1979.

[22] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. `http://www.w3.org/TR/rdf-schema/`, February 2004.

[23] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati. Ontologies and databases: The dl-lite approach. In *Reasoning Web. Semantic Technologies for Information Systems*, pages 255–356. Springer, 2009.

[24] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati. Linking Data to Ontologies: The Description Logic *DL-Lite*$_{\mathcal{A}}$. In B. C. Grau, P. Hitzler, C. Shankey, and E. Wallace, editors, *OWLED*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[25] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Can OWL Model Football Leagues? In C. Golbreich, A. Kalyanpur, and B. Parsia, editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[26] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[27] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Path-Based Identification Constraints in Description Logics. In G. Brewka and J. Lang, editors, *KR*, pages 231–241. AAAI Press, 2008.

[28] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. *Artif. Intell.*, 195:335–360, 2013.

[29] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Identification Constraints and Functional Dependencies in Description Logics. In B. Nebel, editor, *IJCAI*, pages 155–160. Morgan Kaufmann, 2001.

[30] D. Calvanese, M. Ortiz, and M. Simkus. Evolving graph databases under description logic constraints. In T. Eiter, B. Glimm, Y. Kazakov, and M. Krötzsch, editors, *Description Logics*, volume 1014 of *CEUR Workshop Proceedings*, pages 120–131. CEUR-WS.org, 2013.

[31] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[32] E. F. Codd. Further normalization of the data base relational model. *IBM Research Report, San Jose, California*, RJ909, 1971.

[33] E. F. Codd. Normalized data structure: A brief tutorial. In E. F. Codd and A. L. Dean, editors, *SIGFIDET Workshop*, pages 1–17. ACM, 1971.

[34] E. F. Codd. Recent Investigations in Relational Data Base Systems. In *IFIP Congress*, pages 1017–1021, 1974.

[35] H. Darwen, C. J. Date, and R. Fagin. A normal form for preventing redundant tuples in relational databases. In A. Deutsch, editor, *ICDT*, pages 114–126. ACM, 2012.

[36] C. Date and H. Darwen. *Temporal Data and the Relational Model*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[37] D. W. Embley and W. Y. Mok. Developing xml documents with guaranteed "good" properties. In *In ER'01*, pages 426–441, 2001.

[38] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst.*, 2(3):262–278, Sept. 1977.

[39] R. Fagin. Normal forms and relational database operators. In P. A. Bernstein, editor, *SIGMOD Conference*, pages 153–160. ACM, 1979.

[40] R. Fagin. A normal form for relational databases that is based on domains and keys. *ACM Trans. Database Syst.*, 6(3):387–415, Sept. 1981.

[41] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[42] W. Fan and L. Libkin. On xml integrity constraints in the presence of dtds. In P. Buneman, editor, *PODS*. ACM, 2001.

[43] W. Fan and L. Libkin. On xml integrity constraints in the presence of dtds. *J. ACM*, 49(3):368–406, 2002.

[44] P. Hayes. RDF Semantics. `http://www.w3.org/TR/rdf-mt/`, February 2004.

[45] P. Hitzler, M. Krötzsch, and S. Rudolph. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC textbooks in computing series. Chapman & Hall/CRC Press, 2010.

[46] R. Hull and W. Fan, editors. *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*. ACM, 2013.

[47] D. S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and system Sciences*, 28(1):167–189, 1984.

[48] E. Kharlamov. Model theory and calcalus for the description logic dl-lite$_{\mathbb{F}}$. Master's thesis, Free University of Bozen-Bolzano, 2006.

[49] S. Kolahi. Dependency-preserving normalization of relational and xml data. In G. M. Bierman and C. Koch, editors, *DBPL*, volume 3774 of *Lecture Notes in Computer Science*, pages 247–261. Springer, 2005.

[50] H. S. Kunii. Dbms with graph data model for knowledge handling. In *Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*, pages 138–142. IEEE Computer Society Press, 1987.

[51] G. M. Kuper and M. Y. Vardi. A new approach to database logic. In D. J. Rosenkrantz and R. Fagin, editors, *PODS*, pages 86–96. ACM, 1984.

[52] M.-L. Lee, T. W. Ling, and W. L. Low. Designing functional dependencies for xml. In *Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '02, pages 124–141, London, UK, UK, 2002. Springer-Verlag.

[53] M. Lenzerini. Ontology-based data management. In C. Macdonald, I. Ounis, and I. Ruthven, editors, *CIKM*, pages 5–6. ACM, 2011.

[54] M. Ley. DBLP. `http://www.informatik.uni-trier.de/~ley/db/`, 2003.

[55] L. Libkin, J. L. Reutter, and D. Vrgoc. Trial for rdf: adapting graph query languages for rdf data. In Hull and Fan [46], pages 201–212.

[56] H. Mannila and K.-J. Räihä. *The design of relational databases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1992.

[57] W. Y. Mok, Y.-K. Ng, and D. W. Embley. A normal form for precisely characterizing redundancy in nested relations. *ACM Trans. Database Syst.*, 21(1):77–106, 1996.

[58] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language: Profiles. `http://www.w3.org/TR/owl2-profiles/`, December 2012.

[59] B. Motik, P. F. Patel-Schneider, and B. C. Grau. OWL 2 Web Ontology Language: Semantics. `http://www.w3.org/TR/owl2-direct-semantics/`, December 2012.

[60] A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. Linking Data to Ontologies. *J. Data Semantics*, 10:133–173, 2008.

[61] O. Romero, D. Calvanese, A. Abello, and M. Rodriguez-Muro. Discovering functional dependencies for multidimensional design. In *Proc. of the 12th ACM Int. Workshop on Data Warehousing and OLAP (DOLAP 2009)*, pages 1–8, 2009.

[62] D. Sangiorgi. *Introduction to bisimulation and coinduction*. Cambridge University Press, 2012.

[63] J. F. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to RDF and OWL. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 649–658, New York, NY, USA, 2012. ACM.

[64] N. Shadbolt, W. Hall, and T. Berners-Lee. The semantic web revisited. *Intelligent Systems, IEEE*, 21(3):96–101, 2006.

[65] O. W. G. W3C. OWL 2 Web Ontology Language Document Overview (Second Edition). `http://www.w3.org/TR/owl-ref/`, December 2012.

[66] X. C. W. G. W3C. Extensible Markup Language (XML) 1.1 (Second Edition). `http://www.w3.org/TR/xml11/`, August 2006.

[67] X. C. W. G. W3C. Extensible Markup Language (XML) (Fifth Edition). `http://www.w3.org/TR/xml/`, November 2008.

[68] X. S. W. G. W3C. XML Schema 1.1. `http://www.w3.org/XML/Schema`, May 2001.

[69] C. Zaniolo. A new normal form for the design of relational database schemata. *ACM Trans. Database Syst.*, 7(3):489–499, 1982.