DIPLOMARBEIT

Anne Hoffmann

Diplomarbeit

Functional Dependencies in the Higher-Order Entity-Relationship Model

Anne Hoffmann

vorgelegt als Diplomarbeit am
Institut für Informatik der
Technischen Universität Clausthal
Oktober 2002

Erstgutachter:	Prof. Dr. Wilfried Lex
Zweitgutachter:	Prof. Dr. Klaus-Dieter Schewe
Betreuer:	Sebastian Link

Diese Arbeit wurde angefertigt am Department of Information Systems der Massey University in Palmerston North, Neuseeland.

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Palmerston North, den 3. Oktober 2002 _____

Acknowledgement

First of all, I would like to say thank you to my supervisors Klaus-Dieter Schewe and Wilfried Lex for providing me with this interesting topic.

Special thanks to Sebastian Link for numerous interesting discussions on my topic.

Furthermore, I would like to thank all people of the Department of Information Systems at Massey University in Palmerston North in New Zealand for their warm and friendly welcome.

Thank you to all my friends who shared the last couple of years with me.

Special thanks to Stephan Ringel for the time we have shared talking about life, the universe and everything.

Last, but not least thanks to everyone helping me become the person I am.

Experience is one thing
you can't get for nothing.

Oscar Wilde

Zusammenfassung

Die vorliegende Diplomarbeit beschäftigt sich mit der Frage der Axiomatisierbarkeit funktionaler Abhängigkeiten in konzeptionalen Datenbanken. Den Betrachtungen liegt das Higher-Order Entity-Relationship Modell zugrunde. Die Arbeit entstand im Rahmen eines einjährigen Studienaufenthaltes bei Prof. Schewe an der Massey University in Palmerston North in Neuseeland.

Datenbanken wurden entwickelt, um die Verwaltung von Massendaten im Mehrbenutzerbetrieb zu ermöglichen. Bis heute basieren die meisten Datenbanken und deren theoretischen Betrachtungen auf dem von E. F. Codd entwickelten Relationalen Datenmodel (RDM). Die Beschreibungen von Beziehungen zwischen Daten erfolgt im RDM mittels einfacher Relationen. Dies und die Tatsache, daß das RDM für die meisten Anwendungen genügend Struktur bietet haben dazu geführt, daß das RDM das meist verbreitete Datenmodell ist.

Datenbanken modellieren Objekte der realen Welt und ihre Beziehungen untereinander. Abhängigkeiten in Datenbanken definieren dann einfach die Einschränkungen an die Beziehung, in der Objekte einer Datenbank zueinander stehen.

Eine wichtige Klasse von Abhängigkeiten sind funktionale Abhängigkeiten. Wenn wir funktionale Abhängigkeiten betrachten, so betrachten wir Abängigkeiten zwischen Attributmengen eines Relationenschemas. Eine Menge A von Attributen hängt von einer Menge B von Attributen funktional ab, wenn gleiche Einträge in B gleiche Einträge in A implizieren. Daten in Datenbanken sollen möglichst redundanzfrei gespeichert und anomalienfrei verändert werden können. Die Boyce-Codd-Normalform (BCNF) erfüllt diese Forderung für funktionale Abhängigkeiten.

Zur expliziten Bestimmung der BCNF wird für eine Menge vorgegebener Abhängigkeiten die Menge aller folgerbaren Abhängigkeiten benötigt. Ist diese Menge bestimmbar, ohne alle Beziehungen der Datenbank betrachten zu müssen? Unter Verwendung eines korrekten und vollständigen System von syntaktischen Ableitungsregeln, den sogenannten Armstrong-Axiomen, ist es möglich, die Menge aller folgerbaren Abhängigkeiten effektiv zu bestimmen. Ableitungsregeln heißen korrekt, wenn jede ableitbare Abhängigkeit auch logisch folgerbar ist. Sie heißen vollständig, wenn jede folgerbare Abhängigkeit auch syntaktisch ableitbar ist. Eine Klasse von Abhängigkeiten, für die ein endliches korrektes und vollständiges Ableitungssystem existiert, heißt axiomatisierbar.

In den letzten zwanzig Jahren wurden Datenbankschemata von der konzeptionellen Seite

betrachtet. Ein vielzitiertes Modell ist das Entity-Relationship Modell (ERM), das von P. P.-S. Chen eingeführt wurde. Seitdem hat es viele Erweiterungen dieses Modells gegeben. Eine Erweiterung ist das Higher-Order Entity-Relationship Modell (HERM) von B. Thalheim. Im HERM sind Beziehungen auch zwischen Beziehungen selbst erlaubt. So ermöglicht HERM eine sehr realitätsnahe Modellierung. Zudem ist die zugrundeliegende Theorie wohldefiniert. Um eine Normalform ähnlich der BCNF im RDM formulieren zu können, muß zunächst wieder die Frage der Axiomatisierbarkeit beantwortet werden. Erst die Existenz eines Ableitungssystems zur Axiomatisierung funktionaler Abhängigkeiten ermöglicht die explizite Formulierung einer Normalform. Die Betrachtung der Axiomatisierbarkeit funktionaler Abhängigkeiten im HERM ist Gegenstand der vorliegenden Arbeit.

Durch das Erweitern der Beziehungen hat sich das Aussehen der Attribute verändert. Zur Unterscheidung heißen Attribute im RDM einfache oder flache Attribute. Im HERM besteht ein Attribut aus ineinandergeschachtelten Attributen. Diese Eigenschaft ist namensgebend. Attribute heißen im HERM geschachtelte Attribute. Zu jedem Attribut werden Subattribute definiert. Bevor nun die Axiomatisierbarkeit gezeigt werden kann, muß zunächst die Struktur der Menge dieser Subattribute betrachtet werden. Es stellt sich heraus, daß Subattribute im HERM eine Heyting-Algebra bilden. Hierauf aufbauend können die erweiterten Armstrong-Axiome definiert und anschließend das Hauptresultat dieser Arbeit bewiesen werden.

Contents

1 Introduction **3**

1.1 Dependencies in Databases 4

1.2 The Higher-Order Entity-Relationship Model 5

1.3 Functional Dependencies in the HERM 5

1.4 Outline 6

2 Functional Dependencies in the Relational Database Model **7**

2.1 The Relational Database Model 7

2.2 Functional Dependencies in the RDM 11

3 The Higher-Order Entity-Relationship Model (HERM) **19**

3.1 A Type System 20

3.2 Nested Attributes 21

3.3 Entity Types in HERM 23

3.4 Relationship Types in HERM 25

3.5 HERM Schemata 27

4 The Heyting Algebra of Subattributes **31**

4.1 Definitions and Notations 31

4.2 The Lattice of Subattributes 33

4.2.1 Existence of Greatest Lower Bounds 35

4.2.2 Existence of Least Upper Bounds 38

4.3 Distributivity and Relative Pseudo-Complements 40

4.3.1 Distributivity 41

4.3.2 The Pseudo-Complement and the Relative Pseudo-Complement 42

5 Functional Dependencies in HERM **49**

5.1 Functional Dependencies in HERM 49

5.2 Soundness of the Armstrong-axioms 51

5.3 Completeness of the Armstrong-axioms 54

6 Summary **57**

Chapter 1

Introduction

In this thesis we study the axiomatisation of functional dependencies in the Higher-Order Entity-Relationship Model (HERM). What are dependencies? Let us start by giving a short example about data stored about the holdings of a library.

For each book in our library we store certain information. Assume that this information is presented in the form of a table. Each column of the table contains a certain information about the books in the library such as title, author, ISBN, etc. We call the headings of the column attributes. Then every row contains all information about an available book.

Book

Title	Author	ISBN	LibNo	Status	DueDate
A Guide to Latex	H. Kopka	0201398257	777.1	borrowed	27.09.2002
A Guide to Latex	H. Kopka	0201398257	777.2	borrowed	23.07.2002
A Guide to Latex	H. Kopka	0201398257	777.3	available	unknown
The Structure of the RDM	J. Paredaens	3540137149	666.1	borrowed	05.09.2002
The Structure of the RDM	J. Paredaens	3540137149	666.2	lost	unknown
Entity-Relationship Modeling	B. Thalheim	3540654704	555.1	lost	unknown
Entity-Relationship Modeling	B. Thalheim	3540654704	555.2	available	today
Standard C	P.J. Plauger	0134364112	444	available	today
Unix - Network programming	W. R. Stevens	013490012X	333	borrowed	03.10.2002

We need a system where data storage and retrieval is organised independently from programs using the data. Moreover, we want to simplify the updating and storing of data.

The Relational Datamodel (RDM) is a model which provides the formal basis of such systems. It was proposed by E. F. Codd in the early 1970s as a basis for describing the structure and the manipulations of stored data. The RDM is based on the mathematical concept of relations as the only concept, i.e., it uses only relations to descibe the data stored in a database. The simplicity of the RDM helped the RDM to become the most widely used data model. As the data in the RDM are described by rows of tables, it is very easy to understand. In the terminology of databases rows are called tuples. In the RDM

the name of a table together with the names of the columns is called *relation schema*. The names of the columns are called *attributes*. A relation schema together with its contents is called *relation*. Then a *database* is a set of relations.

For instance, we have the relation schema Book with its attributes Title, Author, ISBN, LibNo, Status and DueDate. We write Book = {Title, Author, ISBN, Status, DueDate}.

1.1 Dependencies in Databases

In a database we store properties about real world objects such as books. Database systems have been developed to enable data storage such that multiple user access is possible, and to ensure data integrity. When we work with databases we have to ensure that after storage and manipulation the data still satisfy the integrity constraints in the database. To achieve this goal we consider dependencies.

In the example some attributes determine others, i.e, in our example {ISBN} determines {Title, Author}. This means that in every row the entry in the column ISBN identifies the entry in the columns Title and Author. In other words, the entries in Title and Author depend on the entries in ISBN. This property is expressed by a functional dependency. We write {ISBN}$\rightarrow$ {Title, Author}.

Dependencies express restrictions on the data in a database, e.g., which properties of an object determine other properties of this object. Dependencies may be seen as truth-valued mappings.

Looking at our example we find out that we have several tuples, which have the same entries in {ISBN}. They have the same entries in Title and Author. These data are redundant, as any two tuples coinciding in the ISBN column also coincide in the other two columns Title and Author. So we have data redundancy in our example, because there are tuples, which coincide in {ISBN, Title, Author}.

A database is considered to be well-defined if it does not allow redundant data to be stored. One problem with redundant data is that whenever we insert a new tuple with an ISBN that already exists in the database we have to check, whether it has also the same title and author as all other tuples with this ISBN. Thus, a desired property of a database is the absence of redundancies.

In order to syntactically characterise desirable semantic properties of a database such as the absence of redundancy we must know which dependencies are implied by a given set of dependencies. A dependency d is said to be implied by a set of dependencies dep if each database that satisfies all dependencies in dep also satisfies d. Assume we have a set dep of dependencies given by the user of a database and we want to know the set dep^* of all dependencies implied by dep. This problem is known as the *implication problem* for (functional) dependencies.

As it is hardly feasible to investigate each database, we need to find an axiomatisation, i.e.,

we need to find a system of rules and axioms which allows us to determine the set dep^* of all implied dependencies without using all relations of the database schema.

For the implication of functional dependencies in the Relational Database Model (RDM) W.W. Armstrong showed in [Arm74] that there is a minimal, sound and complete system of rules and axioms. This system is now called the set of *Armstrong-axioms*.

A system of rules and axioms is called *sound*, if all dependencies d, which are derivable from the given set dep of dependencies using the system of rules and axioms, are also implied by the set dep. The system is said to be *complete*, if every dependency that is implied by dep can also be derived with the given system of axioms and rules.

1.2 The Higher-Order Entity-Relationship Model

Since the late eighties several so-called semantic datamodels have been developed, mainly for the purpose of simplifying data modelling and design on a conceptual level closer to the application. The RDM has turned out to be too hard to handle this.

It has turned out that semantic modelling of data provides richer structuring possibilities (see [HuKi87]). Two general philosophies have been adopted in developing semantic models: the aggregate-based and the function-based philosophy. The leading representation of the latter one is the Functional Datamodel (FDM) introduced by D. Shipman in [Shi81]. The former one is the underlying philosophy of the Entity-Relationship Model (ERM), introduced by P. P.-S. Chen in [Che76]. We deal with the aggregate-based philosophy and consider the Higher-Order Entity-Relationship Model (HERM), an advancement of the ERM. The HERM was introduced by Bernhard Thalheim (see [Tha00]). Basic modelling units are entities and relationships. Entities are objects that exist independently from other entities, whereas relationships may be seen as providing connections between objects. Furthermore, every object, i.e., each entity and relationship possesses attributes. Properties of objects are described by values of these attributes.

In contrast to the basic ERM relationships in HERM are not restricted to relationships between entities. Also relationships over relationships can be defined. Such relationships are called higher-level relationships. With higher-level relationships, the HERM provides a sophisticated mechanism to represent complex relations among data. In addition to relationships over relationships the structure of attributes provided in the HERM is more complex. Attributes in HERM can be constructed out of basic simple attributes using constructors such as records and finite sets. We call these attributes *nested attributes*.

1.3 Functional Dependencies in the HERM

In this thesis we consider the generalisation of functional dependencies to the HERM. We want to know, whether there also exists an axiomatisation as in case of the RDM.

For this we look at the structure of the set of subattributes of a nested attribute, as these replace the power set of the set of attributes in the RDM. The completeness proof for the RDM exploits the operations of intersection, union and complement on this power set. We define analogous operations for subattributes. However, it turns out that we only obtain a quasi-Boolean algebra (or Heyting algebra), not a Boolean algebra. In particular, we do not have complements any more as in the RDM.

Instead of providing complements, a Heyting algebra only guarantees the existence of so-called relative pseudo-complements. Nevertheless, it will turn out that relative pseudo-complements will be sufficient to generalise the Armstrong-axioms for functional dependencies from the RDM to the HERM and to prove their soundness and completeness.

1.4 Outline

We start with a brief review of the Relational Database Model (RDM) in Chapter 2. Here we introduce the RDM and explain some basic terminology of databases. Moreover, we present the result of W.W. Armstrong that the Armstrong-axioms form a minimal, sound and complete axiom and rule system for functional dependencies in the RDM.

Then we introduce the Higher-Order Entity Relationship Model (HERM) in Chapter 3, and explain some of its advanced features. We briefly repeat the extended definitions of the basic database terms. We introduce new structures which will be used in the HERM, such as nested attributes, entity and relationship types.

In Chapter 4 we take a closer look at nested attributes, because they define one of the main additional features of HERM. We show that nested attributes form a Heyting Algebra. However, knowing that we obtain a Heyting algebra is not sufficient for our purpose. Therefore, we investigate in detail the structure of the operations in the Heyting algebra of subattributes. In particular, we show how the meet, join, pseudo-complement and relative pseudo-complements look like.

Finally, we present the main result of this thesis in Chapter 5. In this chapter we prove the axiomatisability of functional dependencies in HERM. We conclude this thesis with a summary mainly focussing on open problems regarding dependencies in HERM.

Chapter 2

Functional Dependencies in the Relational Database Model

Our goal is to generalise the finite axiomatisation of functional dependencies from the Relational Database Model (RDM) to the conceptual datamodel HERM. Before we can approach this problem, we briefly summarise the theory of functional dependencies in the RDM, as this will give as a guideline how to approach the generalisation.

The Relational Database Model (RDM) was introduced by E. F. Codd in 1970 [Cod70]. Today most database management systems are based on the RDM and the theory of databases is mainly the theory of the RDM. This has two reasons. First, this theory is elegant and simple, because it is based solely on the mathematical concept of a relation, which immeadiately leads us to first-order logic. The second reason is that the functionality of the RDM is sufficient for most database applications.

Data in a relational database is organised in the form of relations, which can be presented in the form of tables. Each row of such a table defines an object in a database. Columns correspond to properties of objects. Such properties are called attributes.

As objects can be inserted into a database, deleted and modified, the contents of databases vary over time, whereas the database schema, which is given by a set of relation schemata, i.e., sets of attributes, remains unchanged.

2.1 The Relational Database Model

There are two possibilities considering attributes of a relation schemata. One is to define attributes and their domains for each relation schema. This is called the local perspective. The other one is called global perspective. In this case attributes and their domains are defined independently from relation schemata, i.e., the same attributes appearing in different relation schemata correlate to each other. The latter point of view on attributes

simplifies theoretical considerations on databases. Therefore, we start by defining a global universe. Let $\mathcal{D} = \{D_i\}_{i \in I}$ be some family of sets. Each D_i is referred to as a domain. For instance we could have domains such as $\mathcal{A}^*$, $\mathbb{B}$, $\mathbb{N}$. Here $\mathcal{A}^*$ is the set of all character strings over an alphabet $\mathcal{A}$. $\mathbb{B}$ is the set of Boolean truth values, i.e., the set that contains only the two elements 1 and 0, and $\mathbb{N}$ is the set which contains all non-negative integers.

In a universe we consider a set $\mathcal{U}$ of attributes together with a domain assignment *dom* which assigns a domain with each attribute $A \in \mathcal{U}$.

Definition 2.1 (universe). Let $\mathcal{D}$ be a non-empty family of sets. A *universe* $(\mathcal{U}, \mathcal{D} = \{D_i\}_{i \in I}, dom)$ over $\mathcal{D}$ consists of an at most countable set $\mathcal{U}$ and a (global) domain assignment $dom : \mathcal{U} \to \mathcal{D}$. We call the elements of $\mathcal{U}$ *attributes*. □

We are now ready to introduce the definition of a relation schema. A *relation schema* captures the properties that are considered to characterise the objects that should appear in databases.

Definition 2.2 (relation schema R, database schema). Let $(\mathcal{U}, \mathcal{D} = \{D_i\}_{i \in I}, dom)$ be a universe.

1. A *relation schema* R is a finite, non-empty set of attributes , i.e., $R \subseteq \mathcal{U}$ with $\mid R \mid < \infty$ and $R \neq \emptyset$.
2. A *relational database schema* is a finite set of relation schemata.

□

A relation schema R defines the structure of all data that are to be stored over R. In order to insert an object over R, a value from the corresponding domain of every attribute name in R has to be defined.

This means that, in terms of tables if we want to define an object, we define an entry for each column in the same row. In formal terms a row is a tuple and a table is a relation. We now have to define relations for a relation schema. Similary, we have to define databases for a database schema. For this we have to introduce tuples first.

A tuple is a mapping which assigns a value to each attribute. A set of such tuples is called a relation. A database is a family of relations, for the relation schemata in a given database schema.

Definition 2.3 (tuple, relation, database).

1. Let $R = \{A_1, \ldots, A_n\}$ be a relation schema. A *tuple over* R (or an *R-tuple* for short) is a mapping $t : R \to \bigcup_{i \in I} D_i$ with $t(A_i) \in dom(A_i)$ for all $1 \leq i \leq n$. A *relation over* R (or an *R-relation* for short) is a finite set r of R-tuples.

2. If $S = \{R_1, \ldots, R_n\}$ is a database schema, then a *database over* S (or *an* S*-database*) is an S-indexed family $\{r_i\}_{i \in I}$, where each r_i is an R_i-relation for all $1 \leq i \leq n$.

□

An R-relation can be regarded as a relation in usual sense. If an order of the attribute is fixed, an R-tuple tuple turns into a "normal" tuple. Thus, for $R = \{A_1, \ldots, A_n\}$, we obtain $r \subseteq \prod_{i=1}^{n} dom(A_i)$. Let us now look at a simple example for a library database.

Example 2.4. We consider a database schema

Library= {Book, Section, Language, Holder } with the following relation schemata:

Book = { Title, Author, ISBN, LibNo },

Section = { LibNo, Category },

Language = { LibNo, Language }, Holder = { Name, UserId, LibNo },

with $dom(A) = STRING$ for $A \in$ { Title, Author, Name, Category, Language } and $dom(A) = \mathbb{N}$ for $A \in$ { ISBN, LibNo, UserId }.

We now present an example for a database over the database schema in Example 2.4.

Example 2.5. In order to define a database over the schema Library we have to define R-relations for every $R \in$ Library. Thus, we define relations over

Book = { Title, Author, ISBN, LibNo },

Section = { LibNo, Category },

Language = { LibNo, Language } and

Holder = { Name, UserId, LibNo }, respectively:

Book

Title	Author	ISBN	LibNo
A Guide to Latex	H. Kopka	0201398257	777.1
A Guide to Latex	H. Kopka	0201398257	777.2
The Structure of the Relational Database Model	J. Paredaens	3540137149	666
Entity-Relationship Modeling	B. Thalheim	3540654704	555
Standard C	P.J. Plauger	0134364112	444
Unix - Network programming	W. R. Stevens	013490012X	333

Section

LibNo	Category
777.1	Programming
777.2	Programming
666	Data Concepts
555	Data Concepts
444	Programming
333	Network Programming

Language

LibNo	Language
777.1	English
777.2	English
666	English
555	English
444	English
333	English

Holder

Name	UserId	LibNo
Miller	3291	555
Smith	12984	777.1
Wang	4509	444

□

Not all tuples can be used in a relation, i.e., no two books with the same ISBN and two different names for the same UserId, no two books with same LibNo, etc. A combination of attributes, which uniquely identifies tuples, will be called a *key*. In order to introduce keys, we need the notion of a projection.

Definition 2.6 (projection). Let r be a relation over R, and $X \subseteq R$. For a tuple $t \in r$ we define the *projection of* t, denoted $t\ |_X$ as the mapping $t' : X \to \bigcup_{A \in X} dom(A)$ with $t'(A) \in dom(A)$ such that $t'(A) = t(A)$ for each $A \in X$. □

Sometimes the notation $\pi_X(t)$ is used as a synonym t_X.

We now introduce a special set of attributes, called keys. Each tuple of a relation schema is uniquely defined by its keys. A key for which none of its subsets is a key, is called *minimal*.

Definition 2.7 (key, minimal). A *key* on a relation schema R is a subset $K \subseteq attr(R)$ restricting relations r over R to satisfy $t_1 = t_2$ for all tuples $t_1, t_2 \in r$ with $t_1\ |_K = t_2\ |_K$.

A key K is called *minimal* if and only if no proper subset of K is a key. □

We consider a short example for keys in relation schemata.

Example 2.8. We refer to the database schema in Example 2.4 and the database in Example 2.5.
There we had the relation schemata:

Book = { Title, Author, ISBN, LibNo },

Section = { LibNo, Category },

Language = { LibNo, Language },

Holder = { Name, UserId, LibNo }.

For instance, in the relation schema Book the sets {Title, ISBN, LibNo} and {Author, ISBN, LibNo} are keys. But {LibNo} is a minimal key of this relation schema. For the relation schemata Section and Language it is obvious that {LibNo} is key, too. As it is a singleton set, {LibNo} is also the minimal key.

{Name, UserId} is a key for the relation schema Holder, but this is not minimal, as {UserId} is also a key for this relation schema.

Based on the mathematical concept of relations, we found a theory which is easy to understand.

2.2 Functional Dependencies in the RDM

In this section we consider another form of integrity constraints, which generalise keys. In general, integrity constraints are conditions which every instance of a database over a given schema has to satisfy. Thus, integrity constraints restrict the set of databases that are permitted for a schema, and thus can be used to capture more of the semantics of the data.

Some integrity constraints can be formulated by relationships between attributes. The most important class of such constraints is the class of functional dependencies. The importance of functional dependencies is due to the fact that they can be used to describe an easy form of redundancy. As redundancy among the data in a database should be avoided, it is important to understand the theory of functional dependencies.

We will discuss some aspects of the theory of functional dependencies in the RDM. Let X, Y be two sets of attributes of a given relation schema R. We say that Y functionally depends on X iff the projection of tuples to X uniquely determines the projection to Y. In other words, whenever tuples coincide on X, they also coincide on Y. Referring to Example 2.4 we have a functional dependency in the relation schema Book: the attribute set {Author} functionally depends on the set {ISBN}.

Definition 2.9 (functional dependency, satisfaction). Let R be a relation schema. A *functional dependency* on R is an expression of the form $X \to Y$ with $X, Y \subseteq R$.

A relation r over R *satisfies* the functional dependency $X \to Y$ ($\models_r$: $X \to Y$ for short) if and only if $t_1 \mid_X = t_2 \mid_X$ implies $t_1 \mid_Y = t_2 \mid_Y$ for all $t_1, t_2 \in r$. In this case, we call r a *model* for $X \to Y$. □

We now introduce shortcuts in order to simplify the notation of attributes and set of attributes. Usually, with X, Y and Z we denote sets of attributes and A, B and C we denote single attributes. We use juxtaposition to represent the union of attribute sets. Thus, we indentify ABC with the set $\{A, B, C\}$ and XA represents $X \cup \{A\}$.

In the following example we continue Examples 2.4 and 2.5 and consider functional dependencies.

Example 2.10. In the database schema in Example 2.4 we find at least the following two functional dependencies. In the relation schema Book *ISBN* $\rightarrow$ *Author, Title* is a functional dependency and in the relation schema Holder *UserId* $\rightarrow$ *Name* is a functional dependency. The database in Example 2.5 satisfies these functional dependencies.

For instance, ISBN=0201398257 implies Author=H. Kopka and Title=A Guide to Latex. The name of a user is indeed uniquely determined by its UserId. For instance, UserId=777 implies Name=Smith. □

In general, the satisfaction of functional dependencies by a relation r implies the satisfaction of further functional dependencies.

Therefore, we ask how to determine the set of implied functional dependencies. This will first lead to a semantical definition based on the models of a functional dependency, i.e., the relations satisfying it. A functional dependency ψ is implied by a functional dependency ϕ iff all models of ϕ are also models of ψ. A functional dependency ψ is implied by a set Φ of functional dependencies iff each common model of functional dependencies in Φ is also a model of ψ. For each set *dep* of functional dependencies we define its semantic hull *dep** as the set of all those functional dependencies that are implied by *dep*. An obvious disadvantage of this semantic definition is that it involves all R-relations. Therefore, we ask, whether there also exists a set of syntactical derivation rules such that the syntactic hull *dep*$^+$, i.e., the set of all functional dependencies derived from *dep* using these rules, is the same as the semantic hull *dep**. The answer is positive, as well-known result for the implication of functional dependencies showed by W. W. Armstrong in [Arm74].

First we define the notions of implication and semantic hull for functional dependencies.

Definition 2.11 (implication, semantic hull). Let ϕ and ψ be functional dependencies on a relation schema R. We say that ψ is an *implication* of ϕ ($\phi \models \psi$ for short) if and only if each model for ϕ is also a model for ψ.

ψ is an *implication* of a set *dep* of dependencies, if and only if each R-relation r that is a common model of *dep*, i.e., is a model of all $\phi \in$ *dep*, is also a model of ψ (*dep* $\models \psi$ for short).

The *semantic hull dep** of a set *dep* of dependencies is defined as the set of all implied dependencies of *dep* (*dep*$^* = \{\psi \mid$ *dep* $\models \psi\}$). □

In the following, we use parameterised functional dependencies. These are functional dependencies, in which variables may occur. These variables can be used to represent sets of attributes or single attributes. Then $X \rightarrow A$ denote an functional dependency with a set X of attributes and a single attribute A. Knowing that a set of attributes satisfies certain functional dependencies we might be able to derive new functional dependencies. However,

parameters often have to satisfy certain boundary conditions such as being a subset or an element of a certain set of attributes.

For instance, the functional dependency $X \rightarrow A$ holds only if $A \in X$. Then $A \in X$ is called a condition for this functional dependency. Next we introduce derivation rules and axioms.

Definition 2.12 (derivation rule, premises, conclusion, axiom). *A* derivation rule for functional dependencies consists of a finite set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of parameterised functional dependencies, a conclusion $\mathcal{C}$, which is also a parameterised functional dependency, and a finite set *Cond*= $\{C_1, \ldots, C_k\}$ of conditions on the parameters in $\mathcal{P}$ and $\mathcal{C}$. The functional dependencies P_i $(i = 1, \ldots, n)$ are called the *premises* of the rule and the functional dependency $\mathcal{C}$ is called *conclusion* of the rule. A derivation rule without premises $(\mathcal{P} = \emptyset)$ is called an *axiom*. □

Derivation rules are displayed in the following way:

$$\frac{P_1, \ldots, P_n}{\mathcal{C}} C_1, \ldots, C_k.$$

Given a set *dep* of functional dependencies on a relation schema R, we regard these dependencies simply as purely syntactical expressions. Derivation rules then tell us which additional expressions can be derived.

With derivation rules we are able to obtain further functional dependencies by applying the given rules. We look for an instance $\frac{P_1, \ldots, P_n}{\mathcal{C}}$ of a derivation rule , i.e., we replace all variables in $P_1, \ldots, P_n$ and $\mathcal{C}$ such that all conditions $C_1, \ldots, C_k$ are satisfied. Next we define the notion of a derivation tree. A derivation tree represents a finite sequence of applications of rules starting with some given set of functional dependencies.

Definition 2.13 (derivation tree). Let $\mathcal{R}$ be a set of axioms and rules and let *dep* be a set of dependencies. A *derivation tree* over $\mathcal{R}$ and *dep* is a node-labelled tree satisfying the following conditions:

Whenever a node with labelled with the dependency ψ has successor nodes with attached dependencies $\phi_1, \ldots, \phi_n$, then either there exists an instance $\frac{\phi_1, \ldots, \phi_n}{\psi}$ of a rule $\frac{\phi_1', \ldots, \phi_n'}{\psi'} C_1, \ldots, C_k$ in $\mathcal{R}$ or the node is a leaf and $\psi \in$ *dep* holds. □

If ψ is the root of some derivation tree over R and *dep*, then we say that ψ is derivable from *dep*. The syntactic hull is the set of all dependencies that are *derivable from dep*.

Definition 2.14 (derivable, syntactic hull). Le $\mathcal{R}$ be a set of axioms and rules and let *dep* be a set of functional dependencies. A functional dependency is *derivable* from *dep*

using $\mathcal{R}$ if and only if there exists a derivation tree over $\mathcal{R}$ and *dep* with root ψ (notation $dep \vdash_{\mathcal{R}} \psi$).

The *syntactic hull* dep^+ of *dep* under $\mathcal{R}$ is the set of all functional dependencies that are derivable from *dep* using $\mathcal{R}$, i.e., $dep^+ = \{\psi \mid \vdash_{\mathcal{R}} \psi\}$. □

If we deal with a fixed set $\mathcal{R}$ of axioms and rules then we normally drop the index $\mathcal{R}$.

We now give an example for using derivation rules.

Example 2.15. Considering the following three derivation rules:

$$\frac{}{X \to Y} Y \subseteq X \tag{2.1}$$

$$\frac{X \to Y}{X \to XY} \tag{2.2}$$

$$\frac{X \to Y, Y \to Z}{X \to Z} \tag{2.3}$$

where 2.1 is an axiom.

With these inference rules we are able to derive new dependencies from those we had considered in Example 2.5.

1. We take X={*UserId, Name*} and Y=*UserId*. *Using* 2.1 we derive the functional dependency *UserId, Name* → *UserId* , i.e., we used the structure of $\frac{}{X \to Y} Y \subseteq X$.
2. From *UserId* → *LibNo* and *LibNo* → *ISBN*, we derive *UserId* → *ISBN* by applying rule 2.3. We take X=*UserId*, Y=*LibNo and* Z=*ISBN*. Then we use the structure of $\frac{X \to Y, Y \to Z}{X \to Z}$ and infer *UserId*→ *ISBN*.
3. Using 2.2 we can derive the functional dependency *ISBN* → *ISBN, Name* from *ISBN* → *Name*. Take X=*Userid*, Y=*Name*, then use $\frac{X \to Y}{X \to XY}$.

□

We want to relate the semantic and syntactic hulls of a given set of functional dependencies. In fact, we would like to obtain $dep^* = dep^+$ for a suitable set of axioms and rules. A system that allows each derivable functional dependency to be also implied, will be called *sound*. A system that guarantees each implied functional dependency to be also derivable will be called *complete*. Thus, we are looking for a sound and complete set of rules and axioms. If such a set exists, we say that the class of dependencies, i.e., in our case functional dependencies, is *axiomatisable*.

Definition 2.16 (sound, complete, axiomatisable). A set of inference rules $\mathcal{R}$ is said to be *sound* if and only if for each set *dep* of dependencies $dep^+ \subseteq dep^*$ holds.

$\mathcal{R}$ is called *complete* if and only if for all sets *dep* of dependencies $dep^* \subseteq dep^+$ holds.

Any class of dependencies is called *(finitely) axiomatisable* if there exists some finite, sound and complete set of inference rules $\mathcal{R}$ for it. □

Actually, the rules in Example 2.15 are called the Armstrong-axioms for the implication of functional dependencies. Indeed, for this system $dep^* = dep^+$ holds. This means that given a set *dep*, the designer of a database has complete knowledge about dep^*. We now introduce the Armstrong-axioms for functional dependencies formally.

Definition 2.17 (Armstrong-axioms for Functional Dependencies). The *Armstrong-axioms for functional dependencies* are the following three derivation rules:

$$\frac{}{X \to Y} Y \subseteq Y, \qquad \text{(reflexivity rule)}$$

$$\frac{X \to Y, Y \to Z}{X \to Z} \qquad \text{(transitivity rule) and}$$

$$\frac{X \to Y}{X \to XY}. \qquad \text{(extension rule).}$$

□

Example 2.18. We consider a relation schema where letters stand for abstract attribute names.

Let $R = ABCDEF$ be some relation schema with functional dependencies:

$$dep = \{AB \to C, AD \to AB, C \to B, A \to D, CE \to B, BD \to CE, FE \to BA.\}$$

Let $\mathcal{R}$ denote the set of Armstrong-axioms.

1. $\dfrac{BD \to CE, CE \to B}{BD \to B}$ is an simple derivation where we use only the transitivity rule. So $BD \to B \in dep^+$.
2. $\dfrac{CE \to B}{CE \to CEB}$ is another derivation where we use the extension rule only.
3. Now we consider a more complicated derivation.

$$\dfrac{\dfrac{\dfrac{A \to D}{A \to AD} \quad AD \to AB}{A \to AB} \quad \dfrac{AB \to C \quad C \to B}{AB \to B}}{A \to B}$$

So $A \to B \in dep^+$. In particular,

$$\{A \to D, AD \to AB, AB \to C, C \to B\} \vdash_{\mathcal{R}} \{A \to B\}.$$

From soundness, which will be shown in Theorem 2.20, we know:

$$\{A \to D, AD \to AB, AB \to C, C \to B\} \models \{A \to B\}.$$

□

Before we show the main result, we derive some more derivation rules from the Armstrong-axioms.

Proposition 2.19. We can derive the following rules from the Armstrong-axioms:

$$\frac{X \to Y, X \to Z}{X \to YZ} \qquad \textbf{(union rule)}$$

$$\frac{X \to Y, X \to Z}{X \to Y \cap Z} \qquad \textbf{(intersection rule)}$$

$$\frac{X \to Y, Y \to Z}{X \to Z - Y} \qquad \textbf{(complement rule)}$$

$$\frac{X \to Y}{X \to A} A \in Y \qquad \textbf{(fragmentation rule)}$$

$$\frac{X \to Y}{U \to V} X \subseteq U, V \subseteq XY \qquad \textbf{(general extension rule)}$$

$$\frac{X \to Y, U \to V}{W \to Z} U \subseteq XY, X \subseteq W, Z \subseteq VW \qquad \textbf{(general transitivity rule)}$$

□

Proof. We derive the rules.
union rule:

$$\dfrac{\dfrac{X \to Y}{X \to XY} \qquad \dfrac{\dfrac{\overline{XY \to X} \qquad X \to Z}{XY \to Z}}{XY \to XYZ} \qquad \dfrac{}{XYZ \to YZ}}{XY \to YZ}}{X \to YZ}$$

intersection rule:

$$\frac{X \to Y \quad \overline{Y \to Y \cap Z}}{X \to Y \cap Z}$$

complement rule:

$$\frac{X \to Z \quad \overline{Z \to Z - Y}}{X \to Z - Y}$$

fragmentation rule:

$$\frac{X \to Y \quad \overline{Y \to A}}{X \to A}$$

general extension rule:

$$\frac{\dfrac{\overline{U \to X} \quad \dfrac{X \to Y}{\overline{X \to XY}}}{U \to XY} \quad \overline{XY \to V}}{U \to V}$$

general transitivity rule:

$$\frac{\dfrac{\dfrac{\dfrac{\overline{W \to X} \quad \dfrac{X \to Y}{\overline{X \to XY}}}{W \to XY} \quad \overline{XY \to U}}{W \to U} \quad U \to V}{W \to V}}{\dfrac{\overline{W \to VW}}{W \to Z} \qquad \overline{VW \to Z}}$$

□

These rules are used in the proof of the theorem due to Armstrong [Arm74].

Theorem 2.20. *The Armstrong Axioms from Definition 2.17 are a sound and complete set of inference rules for the implication of functional dependencies in the RDM. The set of Armstrong-axioms are minimal in the sense that none of its proper subsets has this property.*

Chapter 3

The Higher-Order Entity-Relationship Model (HERM)

We want to investigate how to generalise functional dependencies to a structually richer data model. In this thesis we are mainly interested in the question, whether such dependencies can be axiomatised.

Many new data models have been introduced over the last two decades. One important class of data models are the so-called conceptual data models. One of the most prominent representative of that class is the Entity-Relationship Model (ERM) developed by P. P.-S. Chen [Che76]. A schema in the ERM describes the real world in terms of entities and relationships between them. Over the years many variations and extensions of the ERM have been developed. For our investigation here we choose the Higher-Order Entity-Relationship Model (HERM) introduced by B. Thalheim [Tha00]. The HERM extends the ERM in several ways.

In the ERM we only used relationships over entities. In HERM we extend these relationships and allow relationships over relationships to be defined as well. For this reason data modelling in HERM is closer to the real world objects. Moreover, HERM features nested attributes using tuple and set constructors.

Further reasons for using HERM are its strict foundation in theory and the easy translation of HERM schemata into RDM schemata. Furthermore, HERM is a data model that can be used as a platform for building database systems.

This chapter summarises the major features of HERM. We briefly repeat fundamental definitions, upon which the generalisation of functional dependencies will be based.

3.1 A Type System

We start with the definition of a universe, i.e., we define a set of (flat) attributes and associated types. Each attribute will be associated with a base type, and to every base type will be assigned a domain.

Definition 3.1 (universe). A *universe* consists of a set $\mathcal{U}$ of attributes, a non-empty family $\mathcal{D} = \{D_i\}_{i \in I}$ of sets, a non-empty set $\mathcal{B}$ of base types and associations *type*: $\mathcal{U} \to \mathcal{B}$ and *dom*: $\mathcal{B} \to \mathcal{D}$. The elements $D_i \in \mathcal{D}$ are called *domains*. □

We write ($\mathcal{U}$, $\mathcal{D}$, $\mathcal{B}$, $type : \mathcal{U} \to \mathcal{B}$, $dom : \mathcal{B} \to \mathcal{D}$) for a universe. In most cases, however, the domains, base types and the association of types to attributes and domains are fixed: so we only write $\mathcal{U}$. We now extend the set $\mathcal{B}$ of base types to a type system $\mathbb{T}$.

Definition 3.2 (type system). The system $\mathbb{T}$ of types is inductively defined as follows:

1. $\mathcal{B} \subseteq \mathbb{T}$
2. $OK \in \mathbb{T}$
3. If $T_1, \ldots, T_n, T \in \mathbb{T}$ and $a_1, \ldots, a_n \in \mathcal{U}$, then $(a_1 : T_1, \ldots, a_n : T_n) \in \mathbb{T}$ and $\{T\} \in \mathbb{T}$.
4. No other types are in $\mathbb{T}$.

□

A type in $\mathcal{B}$ is called a base type. A type of form $(a_1 : T_1, \ldots, a_n : T_n)$ is called a record type with component types $T_1, \ldots, T_n$ and field selectors $a_1, \ldots, a_n$. A type of the form $\{T\}$ is called a set type with component type T.

We want to extend the association of domains to types from base types to the whole type system $\mathbb{T}$. For this we have to extend first the family of domains. $\mathcal{DD}$ is the smallest set extended from $\mathcal{D}$ with the properties as defined below.

Definition 3.3 (constructed domains). The set $\mathcal{DD}$ of *constructed domains* is the smallest set for which the following properties hold:

1. $0, \{ok\} \in \mathcal{DD}$
2. $\mathcal{D} \subseteq \mathcal{DD}$,
3. if $D_1, \ldots, D_n \in \mathcal{DD}$, then $\prod_{i=1}^{n} \{a_i\} \times D_i \in \mathcal{DD}$ with $a_1, \ldots, a_n \in \mathcal{U}$, and
4. if $D \in \mathcal{DD}$, then $\{D' \mid D' \subseteq D, |D'| < \infty\} \in \mathcal{DD}$.

□

We use Definition 3.3 to extend the definition of the domain association $dom : \mathbb{B} \to \mathcal{D}$ to $Dom : \mathbb{T} \to \mathcal{DD}$.

Definition 3.4 (extended domain association Dom). We define the *extended domain assignment* $Dom : \mathbb{T} \to \mathcal{DD}$ as follows.

1. $Dom(OK) := \{ok\}$,
2. $Dom(b) = dom(b)$ for all base types $b \in \mathcal{B}$,
3. $Dom((a_1 : T_1, \ldots, a_n : T_n)) = \prod_{i=1}^{n} \{a_i\} \times Dom(T_i)$ and
4. $Dom(\{T\}) = \mathcal{P}_0(Dom(T))$ where $\mathcal{P}_0(T)$ is the set of all finite subsets of (T).

In the case of a record type $\mathbb{T} = (a_1 : T_1, \ldots, a_n : T_n)$ we write $(a_1 : v_1, \ldots, a_n : v_n)$ with $v_i \in Dom(T_i)$ for a value in $Dom(T)$. We assume that $\mathcal{DD}$ contains at least the set $\{ok\}$. Furthermore, for later use we assume that all domains $D \in \mathcal{DD}$ have at least two elements: $|D| \geq 2$. □

We give a short example for types and their domains.

Example 3.5. We define a type $BOOK$ by

$BOOK$=(Title: {STRING}, Author: {STRING}, ISBN: {NAT}, LibNo: {NAT}).

Assume $dom(STRING) = \mathcal{A}^*$ for the alphabet $\mathcal{A} = \{A, a, B, b, \ldots, Z, z, \_, -\}$ and $dom(NAT) = \mathbb{N}$. Then we get

$$Dom(BOOK)=\{\text{Title: } \mathcal{A}^*, \text{ Author: } \mathcal{A}^*, \text{ ISBN: } \mathbb{N}, \text{ LibNo: } \mathbb{N} \}.$$

3.2 Nested Attributes

Next we extend the set of attributes $\mathcal{U}$ in the given universe to a set $\mathcal{NA}(\mathcal{U})$ of so-called nested attributes. In order to emphasise the distinction from the nested attributes, the attributes in the set $\mathcal{U}$ will be called flat attributes from now on. The use of nested attributes instead of just flat attributes is one of the striking features of HERM.

Definition 3.6 (nested attributes). Let $\mathcal{U}$ be a universe. Let L be an at most countable set of labels disjoint from $\mathcal{U}$ and $\mathcal{D}$. The set of *nested attributes* $\mathcal{NA} = \mathcal{NA}(\mathcal{U})$ is the smallest set with the following properties:

1. $\mathcal{U} \subseteq \mathcal{NA}$,
2. $\lambda \in \mathcal{NA}$,

3. if $X \in L$ and distinct $A_1, \ldots, A_n \in \mathcal{NA}$, then $X(A_1, \ldots, A_n) \in \mathcal{NA}$, and
4. if $X \in L, A \in \mathcal{NA}$ then $X\{A\} \in \mathcal{NA}$ is a finite set attribute.

□

λ is called *null attribute*. An attribute of the form $X(A_1, \ldots, A_n) \in \mathcal{NA}$ is called a record or tuple attribute. An attribute of the form $X\{A\}$ is called a set attribute. Thus, we define a partial order on nested attributes as follows:

Definition 3.7 (partial order on nested attributes). The *partial order* $\leq$ on nested attributes is the smallest order with:

1. $A \leq \lambda$ for all $A \in \mathcal{NA}$
2. $A(A_1, \ldots, A_n) \leq A(A'_{\iota(1)}, \ldots, A'_{\iota(m)})$ with $A_{\iota(i)} \leq A'_{\iota(i)}$ for all $1 \leq i \leq m$ and $\iota : \{1, \ldots, m\} \to \{1, \ldots, n\}$ is monotone and injective.
3. $A\{B\} \leq A\{B'\}$ iff $B \leq B'$.

If $X \leq X'$ holds, we say that X' is a subattribute of X, and X is a superattribute of X'.

□

Next we extend the association of types to attributes from $type : \mathcal{U} \to \mathcal{B}$ to $Type : \mathcal{NA}(\mathcal{U}) \to \mathbb{T}$.

Definition 3.8 (extended type association). The *extended type association* $Type : \mathcal{NA}(\mathcal{U}) \to \mathbb{T}$ is defined as follows:

1. $\text{Type}(\lambda) = OK$
2. $\text{Type}(A) = \text{type}(A)$ for $A \in \mathcal{U}$,
3. $\text{Type}(A(A_1, \ldots, A_n)) = (A_1 : \text{type}(A_1), \ldots, A_n : \text{type}(A_n))$, and
4. $\text{Type}(A\{B\}) = \{\text{type}(B)\}$.

□

The extended type association in Definition 3.8 together with the extended domain association in Definition 3.4 induces a domain assignment Dom on nested attributes with $Dom(X) = Dom(Type(X))$ for all $X \in \mathcal{NA}$. The partial order on nested attributes induces projection functions $\pi^A_{A'}$ for $A \leq A'$ which allows us map the values for some nested attribute A, i.e., $v \in Dom(A)$ to values of a subattribute A', i.e., $\pi^A_{A'}(v) \in Dom(A')$.

Definition 3.9 (projection function). Let A and A' be nested attributes, i.e., $A, A' \in \mathcal{NA}$ with $A \leq A'$. Let t be a value in Dom(A). Then the *projection function* $\pi^A_{A'}$: $Dom(A) \rightarrow Dom(A')$ is defined as follows:

1. $\pi^\lambda_\lambda(t) := id_{\{OK\}}$
2. Assume A is a flat attributes and $A \leq A'$ holds. Then $\pi^A_\lambda(t) := \{ok\}$ for $A' \neq \lambda$, and $\pi^A_{A'}(t) := t$ for $A = A'$.
3. Let $A = A(A_1, \ldots, A_n)$ and $A' = A(A'_{\iota(1)}, \ldots, A'_{\iota(m)}))$ with monotone injective ι : $\{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$. Thus, $A \leq A'$ and $A_{\iota(i)} \leq A'_{\iota(i)}$ for all $i = 1, \ldots, n$ be nested attributes. Then $\pi^A_{A'}(a_1 : t_1, \ldots, a_n : t_n) := A(a_{\iota(1)} : \pi^{A_{\iota(1)}}_{A'_{\iota(1)}}(t_{\iota(1)}), \ldots, a_{\iota(m)} : \pi^{A_{\iota(m)}}_{A'_{\iota(m)}}(t_{\iota(m)}))$ for $i = 1, \ldots, n$.
4. Let $A = A\{B\}$ and $A' = A\{C\}$ be tuple attributes with $B \leq C$. Then $\pi^A_{A'}(\{t_1, \ldots, t_n\}) := A\{\pi^B_C(t_1), \ldots, \pi^B_C(t_n))\}$ holds.

□

A generalised subset of a set of nested attributes is a set Y of nested attributes such that for each element in X a subattribute in Y exists.

Definition 3.10 (generalised subset). Let $X \subseteq \mathcal{NA}$ be a set of nested attributes. $Y = \{A_1, \ldots, A_n\} \subseteq \mathcal{NA}$ is *generalised subset* of X, iff some subset $Z = \{A'_1, \ldots, A'_n\} \subseteq X$ exists with $A'_i \leq A_i$ for $i = 1, \ldots, n$. We write $Y \subseteq_g X$, if Y is a generalised subset of X.

In order to extend the projection function to a generalised subset $Y = \{A_1, \ldots, A_n\}$ of a set $X = \{A_1, \ldots, A_n\}$, we take the projection function and apply it to each element of Y. Then $\pi^X_Y : \prod_{i=1}^{n} Dom(A_i) \rightarrow \prod_{i=1}^{n} Dom(A'_i)$ with $\pi^X_Y(t) := \prod_{i=1}^{n}(\pi^{A_i}_{A'_i}(t))$ defines the projection function on generalised subsets.

3.3 Entity Types in HERM

The idea of an entity is to provide an abstract representation of a real world object through values of some pre-determined attributes. Entities exist independently from other entities. In the HERM the attributes are in fact nested attributes, whereas in the ERM they were only flat attributes. An entity set is a finite set of entities which share the same structure, i.e., the same attributes. An entity type provides the abstract description. Thus, entity types are described by sets of attributes. We denote the set of attributes of an entity type E by $attr(E)$. In addition, the definition of an entity type involves a key. Roughly speaking,

a key provides a way to specify that part of entities must be unique within an entity set. This can be formulised by a generalised subset $id(E) \subseteq attr(E)$. Only those entity sets will be allowed, for which the projection of entities to their keys defines a injective function.

Definition 3.11 (entity type, entity, entity set).

1. An *entity type* $E = (attr(E), id(E))$ consists of
   (a) a finite, non-empty set $attr(E)$ of nested attributes, $attr(E) \subseteq \mathcal{NA}$,
   (b) a primary key $id(E)$, which is a generalised subset of $attr(E)$.
2. An *entity* of type E is a mapping e: $attr(E) \to \bigcup_{A \in attr(E)} Dom(A)$ with $e(A) \in Dom(A)$ for all $A \in attr(E)$.
3. An *entity set* of type E is a finite set E^t of entities of type E with unique key values, i.e., for any two different $e_1, e_2 \in E^t$ we have $\pi_{id(E)}^{attr(E)}(e_1) \neq \pi_{id(E)}^{attr(E)}(e_2)$.

□

We use the following notation. For a tuple of attributes, say $(A_1, \ldots, A_n)$ we write $(A_1 : T_1, \ldots, A_n : T_n)$ which means $type(A_i) = T_i$ for all $i = 1, \ldots, n$. If we assign a value v_i to each attribute A_i, we simply write $(A_1 : v_1, \ldots, A_n : v_n)$. As an example, we take our database from Example 2.4 and write it in the form of an entity type.

Example 3.12. Recall from Example 2.4. We had the database schema of a library Library={Book, Section, Language, Holder}. We obtain

Book = ({Title: $STRING$, Author: $STRING$, ISBNN: NAT, LibNo: NAT},

{LibNo }),

Section = ({LibNo: NAT, Category: $STRING$}, {LibNo}),

Language = ({LibNo: NAT, Language: $STRING$}, {LibNo}) and

Holder = ({Name: $STRING$, UserId: NAT, LibNo: NAT}, {UserId}).

Furthermore, if we consider the instance of our database from Example 2.5, we obtain the same entity sets. Instead of presenting these entity sets, we omit the name and simply list the types.

({Title: A Guide to Latex, Author: H. Kopka, ISBN: 0201398257, LibNo: 777},

{LibNo }),

({Title: The Structure of the RDM, Author: J. Paredaens, ISBN: 3540137149,

LibNo: 666}, {LibNo }),

({Title: Entity-Relationship Modeling, Author: B. Thalheim, ISBN: 3540654704,

LibNo: 555}, {LibNo }),
({Title: Standard C, Author: P.J. Plauger, ISBN: 0134364112, LibNo: 444}, {LibNo }),
({Title: Unix - Network programming, Author: W. R. Stevens, ISBN: 013490012X,
LibNo: 333}, {LibNo }),
({LibNo: 777, Category: Programming}, {LibNo}),
({LibNo: 666, Category: Data Concepts}, {LibNo}),
({LibNo: 555, Category: Data Concepts}, {LibNo}),
({LibNo: 444, Category: Programming}, {LibNo}),
({LibNo: 333, Category: Language}, {LibNo}),
({LibNo: 777, Language: English}, {LibNo}),
({LibNo: 666, Language: English}, {LibNo}),
({LibNo: 555, Language: English}, {LibNo}),
({LibNo: 444, Language: English}, {LibNo}),
({LibNo: 333, Language: English}, {LibNo}),
({Name: Miller, UserId: 3291, LibNo: 555}, {UserId}) ,
({Name: Smith, UserId: 12984, LibNo: 777}, {UserId}) and
({Name: Wang, UserId: 4509, LibNo: 444}, {UserId}).

3.4 Relationship Types in HERM

The idea of a relationship is to provide an abstract description of a real world object, that depends on other real world objects. In addition to using describing attributes we use the objects, on which a relationship depends, as components of the relationship. In the HERM the attributes of relationships are again nested attributes, whereas in the ERM only flat attributes can be used. Furthermore, the components can be relationships, not only entities as in the ERM. However, cycles built by components are excluded.

Analogously to entity sets a relationship set is a set of relationships sharing the same attributes, whilst a relationship type is an abstract description of that structure. Thus, relationship types R are defined by a set $comp(R)$ of components, a set $attr(R)$ of attributes, and a key $id(R)$. In order to let the same entity or relationship type appear more than once as a component of a relationship type R, we use pairwaise different labels — called roles — with the components. In order to exclude cycles we define relationship types of order k for $k \in \mathbb{N}$ and restrict the components to be relationship types of lower order. Then entity types can be considered as relationship types of order 0, i.e., their component sets $comp(E)$ is simply empty.

Definition 3.13 (relationship type, relationship, relationship set).

1. A *relationship type* $R = (comp(R), attr(R), id(R))$ *of order* $i \geq 1$ consists of
   (a) a finite set $comp(R) = \{\xi_1 : R_1, \ldots, \xi_n : R_n\}$ of components with pairs ξ_j: R_j constisting of pairwise different *roles* ξ_j and entity or relationship types R_j of order smaller than i, such that at least one R_j has order exactly $i-1$,
   (b) a finite set $attr(R) \subseteq \mathcal{NA}$ of attributes, and
   (c) a primary $id(R) = comp'(R) \cup id'(R)$ with $comp'(R) \subseteq comp(R)$ and a generalised subset $id'(R)$ of $attr(R)$.
2. Let $Obj(R_i)$ denote the set of all relationships of type R. A *relationship* of type R is a mapping r: $comp(R) \cup attr(E) \rightarrow \bigcup_{i=1}^{n} Obj(R_i) \cup \bigcup_{A \in attr(R)} Dom(A)$ with $r(A) \in Dom(A)$ for all $A \in attr(R)$.
3. A *relationship set of type* R is a finite set R^t of relationship types R with unique key values such that for any two different $r_1, r_2 \in R^t$ we have $\pi_{id(R)}^{comp(R) \cup attr(R)}(r_1) \neq \pi_{id(R)}^{comp(R) \cup attr(R)}(r_2)$.

□

Instead of dealing all the times with the components of relationship types and generalised subsets, it is easier to replace the specification of components and attributes by a simple nested attribute, which of course has the structure of a nested tuple attribute. In particular, the key will turn into a subattribute of this corresponding nested attribute.

Definition 3.14 (corresponding nested attribute).

1. Let E be an entity type with $attr(E) = \{X_1, \ldots, X_n\}$. The *corresponding nested attribute* of E is defined as $N_E := E(X_1, \ldots, X_n)$. The *corresponding key* of E is $K_E = E(X_1', \ldots, X_n')$ for $id(E) = \{X_1', \ldots, X_n'\}$.
2. Let $R = (\{r_1 : R_1, \ldots, r_k : R_k\}, \{X_1, \ldots, X_n\}, id(R))$ be a relationship type with $comp(R) = \{r_1 : R_1, \ldots, r_k : R_k\}$, $attr(R) = \{X_1', \ldots, X_m'\}$ and $id(R) = \{r_{i_1} : R_{i_1}, \ldots, r_{i_l} : R_{i_l}, X_1', \ldots, X_m'\}$. The *corresponding nested attribute* of R is defined as $N_R = R(r_1(N_{R_1}), \ldots, r_l(N_{R_l}), X_1, \ldots, X_n)$. The *corresponding key attribute* is $K_R = R(R_{i1}(N_{R_{i_1}}), \ldots, r_{i_l}(N_{R_{i_l}}), X_1', \ldots, X_m')$.

□

According to the definition of keys we always obtain $N_R \leq K_R$ for all entity and relationship types R. More generally, for each subset $comp'(R) \subseteq comp(R)$ and each generalised subset X of $attr(R)$ we obtain a subattribute of N_R.

Lemma 3.15. *Let* $R = (\{r_1 : R_1, \ldots, r_k : R_k\}, \{X_1, \ldots, X_n\}, id(R))$ *be a relationship type. Then each subattribute of* N_R *is either* λ *or has the form* $R(r_{i_1}(N'_{R_{i_1}}), \ldots, r_{i_l}(N'_{R_{i_l}}), X'_1, \ldots, X'_m)$ *with* $\{i_1, \ldots, i_l\} \subseteq \{1, \ldots, k\}$, $N_{R_i} \leq N'_R$ *and a generalised subset* $\{X'_1, \ldots, X'_m\}$ *of* $attr(R)$. *Conversely, each such pair of a generalised subset of* $comp(R)$ *and a generalised subset of* $attr(R)$ *defines a subattribute of* N_R.

Proof. "$\Longrightarrow$": A generalised subset of $attr(R) = \{X_1, \ldots, X_n\}$ always has the form $Y = \{X'_{\iota(1)}, \ldots, X'_{\iota(m)}\}$ such that $X_{\iota(i)} \leq X'_{\iota(i)}$ holds. Without loss of generality let $\iota(i) = i$ and thus $m \leq n$. As $N_{R_{i_j}} \leq N_{R'_{i_j}}$ holds, the claim follows immediately from the definition of $\leq$.

"$\Longleftarrow$": According to the definition of $\leq$ a subattribute of N_R has the form $R(r_{i_1}(N'_{R_{i_1}}), \ldots, r_{i_l}(N'_{R_{i_l}}), \ldots, X'_{\iota(1)}, \ldots, X'_{\iota(m)})$ with $1 \leq i_1 < \ldots < i_l \leq k$ and $N_{R_{i_j}} \leq N'_{R_{i_j}}$, monotone and injective $\iota : \{1, \ldots, m\} \to \{1, \ldots, n\}$ and $X_{\iota(i)} \leq X'_{\iota(i)}$. Without loss of generality take $\iota(i) = i$, which proves the lemma. □

In the following chapter we will always consider only one nested attribute of a relationship type, the corresponding nested attribute N_R.

3.5 HERM Schemata

We consider a set of entity and relationship types. If this set contains all components of its higher order relationship types, then we call it a HERM schema.

Definition 3.16 (HERM schema). A *HERM schema* is a finite set S of entity and relationship types, such that for every $R \in S$ and for every $r' : R' \in comp(R)$ we have $R' \in S$. □

Given a HERM schema, we look at entity and relationship sets for the entity and relationship types in the schema. If key values are unique and components, entities or relationships exist, we obtain an instance of the HERM schema.

Definition 3.17 (HERM instance). Let S be a HERM schema. An *instance* I *of* S assigns to each entity or relationship type $R \in S$ an entity or relationship set $I(R)$, respectively, such that for all $R \in S$, all $r \in I(R)$ and all $r_i \in R_i \in comp(R)$ the value $v_i = r(r_i : R_i)$ satisfies $v_i \in I(R_i)$. □

Before we conclude this chapter with an example of a HERM schemata we briefly introduce a graphical presentation for such schemata. This graphical presentation is called a HERM diagram.

Definition 3.18 (HERM diagram). A *HERM diagram* for a HERM schema S is a directed, labelled graph $Gr(S)$ defined as follows:

- The vertices of $Gr(S)$ are an one-to-one correspondence to S: each vertex is labelled by an entity or relationship type in S.
- For each component $r_i : R_i \in comp(R)$ there is an edge from R to R_i labelled by r_i.
- Attributes $A \in attr(R)$ are attached to the vertex representing R.

Entity types are usually represented by rectangles, relationship types by diamonds. □

We conclude this chapter with a HERM schema and diagram for our library database. We extend the examples used before such that we receive a HERM schema. Then we present a HERM diagram in Figure 3.1.

Example 3.19. We extend our Example 2.4 such that we obtain higher order schema. Let

Person = ({Name: $STRING$, Surname: $STRING$, Address: $STRING$, Birthday: $Date$, Age: NAT, IdNo: NAT}, {IdNo})

Book = ({Title: $STRING$, Author: $STRING$, ISBN: NAT, LibNo: NAT}, {LibNo}) and

UrgingLetter = ({ExpireDate: $Date$, UrgingLetterNo: NAT, ReminderCharges: NAT},

{UrgingLetterNo})

be relationship types of order 0, i.e., entity types.

Customer = ({is: Person}, {UserId: NAT, Age: NAT, Restriction: $STRING$}, {UserId}),

Copy = ({of: Book}, {CopyNo: NAT, Location: $STRING$}, {CopyNo})

are on level 1 and

Rent = ({heldby: Customer, holds: Copy}, {UserId: NAT, DueDate: $Date$}, {UserId}})

is on level 2. On level 3 we have

Send = ({issend: UrgingLetter, to: Customer}, {SendingDate: $Date$}, {Customer}).

We give an instance of the HERM schema as presented above. Assume we have a database of the library in Palmerston North, New Zealand.

Person

ID	Name	Surname	Address	Birthday	Age	IdNo
P_1	Karl	Smith	7 Victoria Avenue	30.8.1963	39	98547
P_2	Hui	Wang	23 Grey Street	26.04.1952	50	51298
P_3	Todd	Miller	124 Albert Street	12.03.1924	78	75319
P_4	Emelie	Smith	8 Victoria Avenue	23.05.1994	8	455628
P_5	Abbi	McAllen	54 King Street	22.01.1973	29	85462

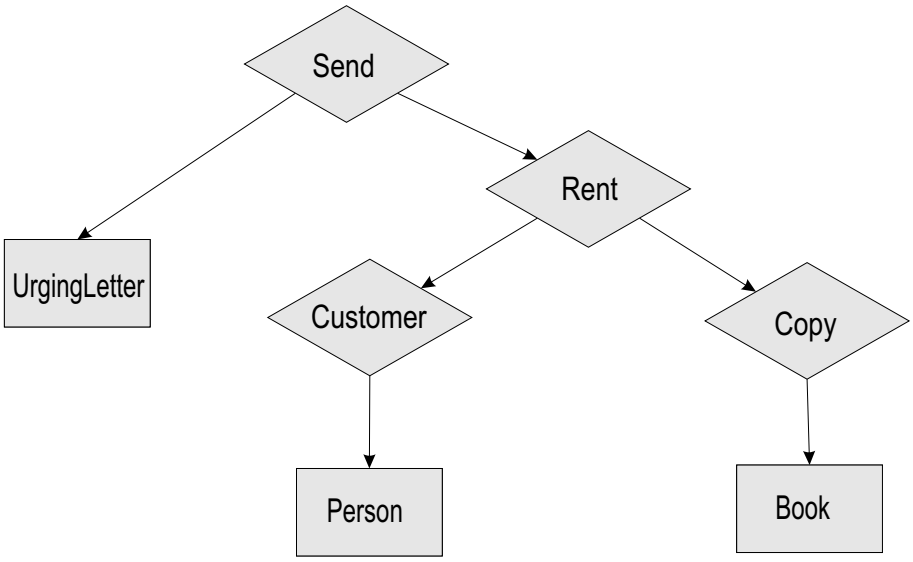

Figure 3.1: Visualisation of HERM example

Book

ID	Title	Author	IsbnNo	LibNo
B_1	A Guide to Latex	H. Kopka	0201398257	777
B_2	The Structure of the RDM	J. Paredaens	3540137149	666
B_3	Entity-Relationship Modeling	B. Thalheim	3540654704	555
B_4	Standard C	P.J. Plauger	0134364112	444
B_5	Unix - Network programming	W. R. Stevens	013490012X	333

UrgingLetter

ID	ExpireDate	UrgingLetterNo	ReminderCharges
UrL_1	23.04.2002	325	12 $

Customer

ID	is	UserId	Age	Restriction
Cu_1	P_1	12984	39	non
Cu_2	P_2	4509	50	non
Cu_3	P_3	3291	78	non
Cu_4	P_4	23127	7	Children books only

Copy

ID	of	CopyNo	Location
Co_1	B_1	1	Programming
Co_2	B_1	2	Programming
Co_3	B_2	1	Data Concepts
Co_4	B_3	1	Data Concepts
Co_5	B_4	1	Programming
Co_6	B_5	1	Language

Rent

heldby	holds	UserId	DueDate
Cu_1	Co_2	12984	04.09.2002
Cu_2	Co_5	4509	23.04.2002
Cu_3	Co_4	3291	15.09.2002

Send

issend	to	SendingDate
Url_1	Cu_2	27.08.2002

Chapter 4

The Heyting Algebra of Subattributes

In this chapter we investigate the structure of the set of subattributes of a given attribute. We will not obtain a Boolean Algebra as in case of the RDM. However, we will find out that the set of subattributes carries the structure of a Heyting algebra.

4.1 Definitions and Notations

We start repeating several definitions from lattice theory and introduce some fundamental notation. The most fundamental definition is the one of a lattice. According to [EoI00, p. 124] a lattice is defined as follows.

Definition 4.1 (lattice). A *lattice* is a structure $(M, \cap, \cup)$ where $\cap$ and $\cup$ are binary functions called *meet* and *join* satisfying for all $a, b, c \in M$:

$$a \cup a = a \cap a = a$$
$$a \cap b = b \cap a,\ a \cup b = b \cup a$$
$$a \cap (b \cap c) = (a \cap b) \cap c,\ a \cup (b \cup c) = (a \cup b) \cup c$$
$$a \cap (a \cup b) = a \cup (a \cap b) = a.$$

□

Definition 4.2 ((least) upper bound, (greatest) lower bound). An *upper bound* of a subset X of a partially ordered set M is an element $a \in M$ which is larger than every $x \in X$. A *least upper bound* is an upper bound smaller than every other upper bounds. The notions of a lower bound and a greatest lower bound are defined dually.

In fact, if we have a partially ordered set $(M, \leq)$ such that for any two elements the least upper bound and the greatest lower bound exist, then M defines a lattice.

Proposition 4.3. *If $\leq$ is partial order on a set M such that for any $a, b \in M$, the greatest lower bound and least upper bound of $\{a, b\}$ exist, denoted by $A \cup B$ and $A \cap B$ respectively, then $(M, \cap, \cup)$ is a lattice.* □

Definition 4.4 (bottom element, top element). Let M be a partially ordered set. If $a \in M$ is the smallest element of M, i.e., $a \leq b$ for all $b \in M$, then we call a the *bottom element* of M, denoted as 0.

Analogously, if $c \in M$ is the greatest element of M, i.e., $c \geq b$ for all $b \in M$, then we call c the *top element*, denoted as 1.

A lattice M with a bottom element may have pseudo-complements for the elements a of M. If such pseudo-complements exist for all $a \in M$, we call the lattice a pseudo-complemented lattice. According to [EoI00, p.179] we define the pseudo-complement as follows.

Definition 4.5 (pseudo-complement, pseudo-complemented lattice). If $(M, \leq)$ is a lattice with a bottom element 0 and $a \in M$, then $b \in M$ is called the *pseudo-complement* of a iff b is the least upper bound of the elements of L disjoint from a, i.e., b is the least upper bound of the set $\{x \in M \mid a \cap x = 0\}$.

If every member of M has a pseudo-complement, $(M, \leq)$ is called a *pseudo-complemented lattice*.

We may generalise this definition, which we may consider defining the pseudo-complement of a is relative to 0. There are lattices where we have pseudo-complements relative to all other elements of the lattice. Such lattices are called Heyting algebras. In order to define a Heyting algebra we have to define the pseudo-complement of a relative to b where a and b are any elements of the lattice M.

Definition 4.6 (distributive lattice). A lattice M is called *distributive* if and only if, for every $a, b, c \in M$

$$a \cap (b \cup c) = (a \cap b) \cup (a \cap c) \text{ and } a \cup (b \cap c) = (a \cup b) \cap (a \cup c).$$

Definition 4.7 (relative pseudo-complement). Suppose $(M, \cup, \cap)$ is a lattice and $a, b \in M$. The *pseudo-complement of a relative to b* (denoted as $a \Longrightarrow b$), is the least upper bound of the set $c = sup\{x \in M \mid a \leq x = b\}$, equivalently, $c \leq (a \Longrightarrow b)$ iff $c \cap a \leq b$. □

Now we can define the notion of a Heyting algebra.

Definition 4.8 (Heyting algebra). A *Heyting algebra* is a distributive lattice $(M, \cap, \cup)$ with bottom element 0 such that for all $a, b \in M$ the pseudo-complement $a \Longrightarrow b$ of a relative to b exists, i.e., $c \leq (a \Longrightarrow b)$ iff $a \cap c \leq b$ holds for all $c \in M$. □

Lemma 4.9. *Let $(M, \cap, \cup)$ be a lattice with a pseudo-complement $a \Longrightarrow b$ of a relative to b. Then the pseudo-complement is unique.*

Proof. Assume, $a \to b$ is another relative pseudo-complement. Then we have to show $(a \to b) \leq (a \Longrightarrow b)$. For similarity reasons the uniqueness follows.

As $\leq$ is reflexive, we obtain $(a \to b) \leq (a \to b)$, which is equivalent to $a \cap (a \to b) \leq b$ according to the definition of relative pseudo-complements. As $a \Longrightarrow b$ is also a relative pseudo-complement, the definition implies again $(a \to b) \leq (a \Longrightarrow b)$. □

Special Heyting algebras are the so-called Boolean algebras. According to [Bir48, p.152] a Boolean algebra is defined as follows.

Definition 4.10 (Boolean algebra). A *Boolean algebra* is a distributive lattice $(M, \cap, \cup)$ such that for each $a \in M$ there exists an element $\overline{a} \in M$ such that for all $a, b \in M$ hold:

1. $\overline{(a \cup b)} = \overline{a} \cap \overline{b}$ and $\overline{(a \cap b)} = \overline{a} \cup \overline{b}$.
2. $\overline{(\overline{a})} = a$
3. $a \cup \overline{a} = 1$ and $a \cap \overline{a} = 0$ where 0 is bottom and 1 top element of the lattice. $\overline{a}$ is called the *complement* of a.

□

Examples of Boolean algebras are power sets with the usual intersection and union operstions. Such a power set algebra is used in the study of functional dependencies in the RDM. For the HERM, however, we will see that Boolean algebras are not sufficient.

4.2 The Lattice of Subattributes

First we define the set of subattributes of a given nested attribute $X \in \mathcal{NA}$. This is the set SubAttr(X)=$\{Y \mid X \leq Y\}$. To simplify our proofs we define an equivalence relation $\equiv$ on SubAttr(X) such that equivalent subattributes have the same domain.

Definition 4.11. Let $X \in \mathcal{NA}$. We define $\equiv \subseteq SubAttr(X) \times SubAttr(X)$ as the smallest equivalence relation on SubAttr(X) with

1. $A(A_1, \ldots A_{i-1}, \lambda, A_i, \ldots A_n) \equiv A(A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n)$,
2. $A(\lambda) \equiv \lambda$ and
3. $A\{B\} \equiv A\{C\}$ if and only if $B \equiv C$.

□

With this definition we are able to restrict our attention to subattributes of the same length, as we may always add or remove λ's. Indeed, we study $\mathrm{SubAttr}(X)/\equiv$, but to simplify the notation we identify $\mathrm{SubAttr}(X)/\equiv$ with $\mathrm{SubAttr}(X)$.

In the following we fix a nested attribute X. We want to show that $\mathrm{SubAttr}(X)$ carries the structure of a Heyting algebra. The partial order $\leq$ on $\mathrm{SubAttr}(X)$ is simply defined by $Y \leq Z$ iff Z is a subattribute of Y ($Y, Z \in \mathrm{SubAttr}(X)$).

With respect to the more familiar database notation the greatest lower bound in this Heyting algebra will be denoted by $\bowtie_X$, the least upper bound by $\bullet_X$.

We first show that $\mathrm{SubAttr}(X)$ is indeed a partially ordered set (poset) with bottom element X and top element λ. As $X \leq Y$ holds by definition for all $Y \in \mathrm{SubAttr}(X)$ and $Y \leq \lambda$ holds by definition of subattributes (see Definition 3.6), the assertions about the top and the bottom element are obvious.

Proposition 4.12. *Let $X \in \mathcal{NA}$. Then (SubAttr(X), $\leq$) is a poset.* □

Proof. We show that $\leq$ is reflexive, transitive and antisymmetric.

1. First we have to show that $Y \leq Y$ holds for all $Y \in \mathrm{SubAttr}(X)$. We use structural induction on X. This is obvious for $Y = \lambda$, so we may assume $Y \neq \lambda$.

   (a) Suppose $X = A \in \mathcal{NA}$ is a flat attribute. Then the subattribute Y is $Y = A$, which implies $A \leq A$.

   (b) Assume X is a tuple attribute, i.e., $X = A(A_1, \ldots, A_n)$. Let Y be any subattribute of X, i.e., $X \leq Y$. According to the definition of the equivalence relation $\equiv$ we may assume $Y = A(A_1', \ldots, A_n')$ with $A_i \leq A_i'$ for all $i = 1, \ldots, n$. By induction we have $A_i' \leq A_i'$ for all $i = 1, \ldots, n$, thus also $Y \leq Y$ by the definition of $\leq$.

   (c) Finally, $X = A\{D\}$ with $D \in \mathcal{NA}$ and $X \leq Y$. Again using $\equiv$ we may assume $Y = A\{C\}$ with $D \leq C$. By induction we have $C \leq C$ and therefore $Y \leq Y$, which completes the proof of the reflexivity axiom.

2. Now we show that $\leq$ is transitive, i.e., for all $U, Y, Z \in \mathrm{SubAttr}(X)$ with $U \leq Y$ and $Y \leq Z$ we obtain $U \leq Z$. We use again structural induction on X. Again, if one of U, Y or Z is λ the result is obvious. Thus, we now assume that all these nested attributes differ from λ.

   (a) Suppose $X = A$ is a flat attribute. In this case we have $U = Y = Z = A$, from which the claim follows immediately.

   (b) Let X be a tuple attribute, i.e., $X = A(A_1, \ldots, A_n)$. Let U be any subattribute of X, i.e., $X \leq U$ and Y any subattribute of U, i.e., $U \leq Y$. Furthermore, be Z any subattribute of Y, i.e., $Y \leq Z$.

According to the definition of the equivalence relation $\equiv$ we may assume $U = A(A_1', \ldots, A_n')$, $Y = A(A_1'', \ldots, A_n'')$ and $Z = A(A_1''', \ldots, A_n''')$ with $A_j \leq A_j'$, $A_j' \leq A_j''$ and $A_j'' \leq A_j'''$ for all $i = 1, \ldots, n$. By induction we have $A_j' \leq A_j'''$ for all $i = 1, \ldots, n$, thus also $U \leq Z$ by the definition of $\leq$.

(c) Finally, $X = A\{D\}$ with $D \in \mathcal{NA}$ and $X \leq U$. Again using $\equiv$ we may assume $U = A\{E\}$ with $D \leq E$. Furthermore, we may assume $Y = A\{C\}$ with $E \leq C$ and $Z = A\{B\}$ with $C \leq B$. By induction we have $E \leq B$ and therefore $A\{E\} \leq A\{B\}$, which proves the transitivity.

3. It remains to verify that $\leq$ is antisymmetric, i.e., for all $Y, Z \in \text{SubAttr}(X)$ with $Y \leq Z$ and $Z \leq Y$ we obtain $Y = Z$. We use again structural induction on X. We may exclude that one of Y or Z is λ, as in this case the result is obvious.

   (a) If $X = A$ is a flat attribute, then $Y = Z = \lambda$ follows.

   (b) Assume X is a tuple attribute, $X = A(A_1, \ldots, A_n)$ with $X \leq Y, Z$. According to the equivalence relation we may assume $Y = A(A_1', \ldots, A_n')$ and $Z = A(A_1'', \ldots, A_n'')$ with $A_i'' \leq A_i'$ and $A_i' \leq A_i''$ for all $i = 1, \ldots, n$. By induction we have $A_i' = A_i''$ for all $i = 1, \ldots, n$ and thus also $Y = Z$.

   (c) Finally, let $X = A\{D\}$ with $D \in \mathcal{NA}$. Let Y, Z be any subattributes of X and $Y \leq Z$. Again using $\equiv$ we may assume $Y = A\{C\}$ and $Z = A\{B\}$ with $D \leq C$, $C \leq B$ and $B \leq C$. By induction we have $C = B$ and therefore $A\{C\} = A\{B\}$. That proves the antisymmetry and concludes the proof of $(\text{SubAttr}(X), \leq)$ being a poset.

□

4.2.1 Existence of Greatest Lower Bounds

We now introduce an operation $\bowtie_X$ on $\text{SubAttr}(X)$. It will turn out that $Y \bowtie_X Z$ is the greatest lower bound (g.l.b.) of subattributes $Y, Z \in \text{SubAttr}(X)$ with respect to $\leq$. In lattice terminology the g.l.b. is also called meet.

Definition 4.13 ($\bowtie_X$). Let X, Y, Z be nested attributes with $X \leq Y, Z$. Then the *meet* $Y \bowtie_X Z$ is inductively defined as follows.

1. Let $X = \lambda$. Then $Y = Z = \lambda$ and we define $Y \bowtie_X Z := \lambda$.

2. Let $X = A$ be a flat attribute. Then

$$Y \bowtie_X Z := \begin{cases} \lambda & \text{if } Y = Z = \lambda \\ A & \text{else} \end{cases}$$

3. Let $X = A(A_1, \ldots, A_n)$, be a nested tuple attribute. Then we may assume $Y = A(A_1', \ldots, A_n')$ and $Z = A(A_1'', \ldots, A_n'')$ with $A_i \leq A_i', A_i''$ for $1 \leq i \leq n$. So we define

$$Y \bowtie_X Z := A(A_1' \bowtie_{A_1} A_1'', \ldots, A_n' \bowtie_{A_n} A_n'').$$

4. Assume now that $X = A\{D\}$ is a nested set attribute. Then we may assume $Y = A\{C\}$ and $Z = A\{B\}$ where $D \leq B, C$. We define

$$A\{C\} \bowtie_X A\{B\} := A\{B \bowtie_D C\}.$$

5. Let X be a nested attribute and assume that at least one of Y or Z (maybe both) is λ, say $Z = \lambda$. Then define $Y \bowtie_X Z = Y$.

□

We have to prove that $\bowtie_X$ defines indeed the meet on the structure (SubAttr(X),$\leq$). This is done in the following proposition.

Proposition 4.14. *Let $X \in \mathcal{NA}$ and $Y, Z \in$ SubAttr(X). Then $Y \bowtie_X Z$ is the meet of Y and Z in (SubAttr(X), $\leq$).* □

Proof. We have to show that $Y \bowtie_X Z \leq Y, Z$ and that for each subattribute X' with $X' \leq Y$ and $X' \leq Z$ also $X' \leq Y \bowtie_X Z$ holds. We use structural induction on X. If at least one of Y, Z is λ, say $Z = \lambda$, then Definition 4.13 gives $Y \bowtie_X Z = Y$. Then the claimed assertions obviously hold. So, in the structural induction we may restrict to both Y and Z being different from λ.

1. Let $X = \lambda$ and $X \leq Y, Z$, i.e., $Y = Z = \lambda$. This case has already been dealt with.

2. Suppose X is a flat attribute, say $X = A$. Then we get $Y = Z = A$ which obviously implies the claim.

3. Let X be a nested record attribute, $X = A(A_1, \ldots, A_n)$, $1 \leq i \leq n$. For $X \leq Y, Z$ we can take $Y = A(A_1', \ldots, A_n')$ and $Z = A(A_1'', \ldots, A_n'')$, $A_j \leq A_j', A_j''$ for $1 \leq j \leq n$. By definition we have

$$Y \bowtie_X Z = A(A_1' \bowtie_{A_1} A_1'', \ldots, A_n' \bowtie_{A_n} A_n'').$$

   By induction we have $A_i' \bowtie_{A_i} A_i'' \leq A_i'$, $A_i' \bowtie_{A_i} A_i'' \leq A_i''$, and each B_i with $B_i \leq A_i'$ and $B_i \leq A_i''$ implies $B_i \leq A_i' \bowtie_{A_i} A_i''$, for all $1 \leq i \leq n$, $Y = A(A_1', \ldots, A_n')$ is subattribute of $Y \bowtie_X Z$. Hence, $Y \bowtie_X Z \leq A(A_1', \ldots, A_n') = Y$ and $Y \bowtie_X Z \leq Z$. Now, let $X' \in$ SubAttr(X). We may assume $X' = A(B_1, \ldots, B_n)$ with $A_i \leq B_i$ for all $i = 1, \ldots, n$. Assuming $X' \leq Y$ and $X' \leq Z$ implies $B_i \leq A_i'$ and $B_i \leq A_i''$ for all $i = 1, \ldots, n$. By induction it follows that $B_i \leq A_i' \bowtie_{A_i} A_i''$ holds for all $i = 1, \ldots, n$. Hence also $X' \leq Y \bowtie_X Z$.

4. Suppose now $X = A\{D\}$ and $X \leq Y, Z$. So we can take $Y = A\{C\}$, $Z = A\{B\}$ with $D \leq B, C$. By induction, we obtain $B \bowtie_D C \leq B$ and $B \bowtie_D C \leq C$, which imply

$$\begin{aligned} Y \bowtie_X Z = A\{B \bowtie_D C\} \leq A\{B\} = Z \\ \text{and } Y \bowtie_X Z = A\{B \bowtie_D C\} \leq A\{C\} = Y. \end{aligned}$$

   For $X' = A\{D'\} \in \text{SubAttr}(X)$, i.e., $D \leq D'$, with $X' \leq Y$ and $X' \leq Z$, we get $D' \leq C \bowtie_D B$ by induction, have also $X' = A\{D'\} \leq A\{C \bowtie_D B\} = Y \bowtie_X Z$, which completes the proof.

□

To get used to the notation we consider a first small example of $\bowtie_X$.

Example 4.15. Assume

$$\begin{aligned} X &= A(A_1(B_1, B_2, B_3\{C_1\}), A_2, A_3(B_4\{(C_2, C_3)\}, B_5), A_4\{(B_6, B_7\{C_4\{D\}\})\}) \\ Y &= A(A_1(B_2, B_3\{\lambda\}), A_3(B_4\{C_2\}, B_5), A_4\{(B_7\{C_4\{D\}\})\}) \\ Z &= A(A_1(B_1, B_2), A_2, A_3(B_4\{C_3\}, B_5), A_4\{(B_6)\}) \end{aligned}$$

We identify Y with

$$Y = A(A_1(\lambda, B_2, B_3\{\lambda\}), \lambda, A_3(B_4\{(C_2, \lambda)\}, B_5), A_4\{(\lambda, B_7\{C_4\{D\}\})\})$$

using the equivalence relation $\equiv$ on $\text{SubAttr}(X)$. Analogously, we identify Z with

$$Z = A(A_1(B_1, B_2, \lambda), A_2, A_3(B_4\{(\lambda, C_3)\}, B_5), A_4\{(B_6, \lambda)\}).$$

For the meet $Y \bowtie_X Z$ we get:

$$\begin{aligned} Y \bowtie_X Z &= A(A_1(\lambda, B_2, B_3\{\lambda\}) \bowtie_{A_1(B_1,B_2,B_3\{C_1\})} A_1(B_1, B_2, \lambda), \\ &\quad \lambda \bowtie_{A_2} A_2, A_3(B_4\{(C_2, \lambda)\}, B_5) \bowtie_{A_3(B_4\{(C_2,C_3)\},B_5)} A_3(B_4\{(\lambda, C_3)\}, B_5), \\ &\quad A_4\{(\lambda, B_7\{C_4\{D\}\})\} \bowtie_{A_4\{(B_6,\lambda)\}} A_4\{(B_6, \lambda)\}) \\ &= A(A_1(\lambda \bowtie_{B_1} B_1, B_2 \bowtie_{B_2} B_2, B_3\{\lambda\} \bowtie_{B_3\{\lambda\}} \lambda), \\ &\quad \lambda \bowtie_{A_2} A_2, A_3(B_4\{(C_2, \lambda)\}) \bowtie_{B_4\{(C_2,C_3)\}} B_4\{(\lambda, C_3)\}, B_5 \bowtie_{B_5} B_5), \\ &\quad A_4\{(\lambda \bowtie_{B_6} B_6, B_7\{C_4\{D\}\})\} \bowtie_{B_7\{C_4\{D\}\}} \lambda) \\ &= A(A_1(B_1, B_2, B_3\{\lambda\}), A_2, A_3(B_4\{(C_2, C_3)\}, B_5), A_4\{(B_6, B_7\{C_4\{D\}\})\}) \end{aligned}$$

Finally, we may identify $Y \bowtie_X Z$ with

$$Y \bowtie_X Z = A(A_1(B_1, B_2, B_3\{\lambda\}), A_2, A_3(B_4\{(C_2, C_3)\}, B_5), A_4\{(B_6, B_7\{C_4\{D\}\})\}).$$

□

4.2.2 Existence of Least Upper Bounds

Analogously to the proof of the existence of greatest lower bounds (g.l.b.) in the previous section we now define an operation $\bullet_X$ on SubAttr(X) and show that $Y \bullet_X Z$ gives us the least upper bound (l.u.b.) — or join — of the subattributes Y, Z in (SubAttr(X), $\leq$). This operation is defined as follows.

Definition 4.16 ($\bullet_X$). The *join* $Y \bowtie_X Z$ between subattributes Y, Z with $X \leq Y, Z$ is inductively defined as follows.

1. If $X = \lambda$, then
$$Y \bullet_X Z = \lambda.$$

2. If $X = A$ is a flat attribute, then
$$Y \bullet_X Z := \begin{cases} A & \text{if } Y = Z = A \\ \lambda & \text{else} \end{cases}$$

3. If $X = A(A_1, \ldots, A_n)$ is a record attribute, $Y = A(A_1', \ldots, A_n') \neq \lambda$ and $Z = A(A_1'', \ldots, A_n'') \neq \lambda$ with $A_i \leq A_i', A_i''$ for all $1 \leq i \leq n$, then
$$Y \bullet_X Z := A(A_1' \bullet_{A_1} A_1'', \ldots, A_n' \bullet_{A_n} A_n'').$$

4. If $X = A\{D\}$ is a set attribute, $Y = A\{B\} \neq \lambda$ and $Z = A\{C\} \neq \lambda$ with $D \leq B, C$, then
$$Y \bullet_X Z := A\{B \bullet_D C\}.$$

5. If X is a nested record or set attribute and one of Y or Z is λ, then $Y \bullet_X Z = \lambda$.

□

Analogously to Proposition 4.14 we have to show that the operation $\bullet_X$ as defined above defines indeed the least upper bound in (SubAttr(X),$\leq$).

Proposition 4.17. *Let $X \in \mathcal{NA}$ and $Y, Z \in$ SubAttr(X). Then $Y \bullet_X Z$ is the join of Y and Z in (SubAttr(X)).* □

Proof. We have to show that $Y \leq Y \bullet_X Z$ and $Z \leq Y \bullet_X Z$, and that for each subattribute X' with $Y \leq X'$ and $Z \leq X'$ also $Y \bullet_X Z \leq X'$ holds. We use structural induction on X. If at least one of Y, Z is λ, say $Z = \lambda$, then Definition gives $Y \bullet_X Z = \lambda$. So without loss of generality assume that both Y and Z are different from λ.

1. Let $X = \lambda$ and $X \leq Y, Z$, i.e., $Y = Z = \lambda$. This case has already been dealt with.

2. Suppose X is a flat attribute, say $X = A$. Then we get $Y = Z = A$ which obviously implies the claim.

3. Let X be a record attribute. Assume $X = A(A_1, \ldots, A_n)$, $1 \leq i \leq n$. For $X \leq Y, Z$ we can take $Y = A(A_1', \ldots, A_n')$ and $Z = A(A_1'', \ldots, A_n'')$ with $A_j \leq A_j', A_j''$ for $1 \leq j \leq n$. By definitition we have

$$Y \bullet_X Z = A(A_1' \bullet_{A_1} A_1'', \ldots, A_n' \bullet_{A_n} A_n'').$$

   By induction we have $A_j' \leq A_j' \bullet_{A_j} A_j''$ and $A_j'' \leq A_j' \bullet_{A_j} A_j''$ for all $i = 1, \ldots, n$. Each B_j with $A_j' \leq B_j$ and $A_j'' \leq B_j$ implies $A_i' \bullet_{A_j} A_j'' \leq B_j$ for all $1 \leq j \leq n$, $Y = A(A_1', \ldots, A_n')$ is superattribute of $Y \bullet_X Z$.

   Hence $Y = A(A_1', \ldots, A_n') \leq Y \bullet_X Z$ and similarly $Z \leq Y \bullet_X Z$.

   Now let $X' \in \mathrm{SubAttr}(X)$. We may assume $X' = A(B_1, \ldots, B_n)$ with $A_i \leq B_i$ for all $i = 1, \ldots, n$. By induction it follows that $A_i' \bullet_{A_i} A_i'' \leq B_i$ holds for all $i = 1, \ldots, n$. Hence $Y \bullet_X Z \leq X'$.

4. Finally, we have $X = A\{D\}$ and $X \leq Y, Z$. So we can take $Y = A\{B\}$ and $Z = A\{C\}$ wit $D \leq B, C$. By induction we obtain $B \leq B \bullet_X C$ and $C \leq B \bullet_X C$, then

$$\begin{aligned} Y &= A\{B\} \leq A\{B \bullet_D C\} &= Y \bullet_X Z \\ \text{and } Z &= A\{C\} \leq A\{B \bullet_D C\} &= Y \bullet_X Z. \end{aligned}$$

□

We give a simple example for the join operation $\bullet_X$.

Example 4.18. We use the same subattributes as in Example 4.15, i.e.,

$$\begin{aligned} X &= A(A_1(B_1, B_2, B_3\{C_1\}), A_2, A_3(B_4\{(C_2, C_3)\}, B_5), A_4\{(B_6, B_7\{C_4\{D\})\}) \\ Y &= A(A_1(B_2, B_3\{\lambda\}), A_3(B_4\{C_2\}, B_5), A_4\{(B_7\{C_4\{D\}\})\}) \\ Z &= A(A_1(B_1, B_2), A_2, A_3(B_4\{C_3\}, B_5), A_4\{(B_6)\}) \end{aligned}$$

Again we identify Y with

$$Y = A(A_1(\lambda, B_2, B_3\{\lambda\}), \lambda, A_3(B_4\{(C_2, \lambda)\}, B_5), A_4\{(\lambda, B_7\{C_4\{D\}\})\}))$$

using the equivalence relation $\equiv$ on $\mathrm{SubAttr}(X)$. Analogously, we identify Z with

$$Z = A(A_1(B_1, B_2, \lambda), A_2, A_3(B_4\{(\lambda, C_3)\}, B_5), A_4\{(B_6, \lambda)\}).$$

For the join $Y \bullet_X Z$ we get:

$$\begin{aligned} Y \bullet_X Z &= A(A_1(\lambda, B_2, B_3\{\lambda\}) \bullet_{A_1(B_1,B_2,B_3\{C_3\})} A_1(B_1, B_2, \lambda), \lambda \bullet_{A_2} A_2, \\ &\quad A_3(B_4\{(C_2, \lambda)\}, B_5) \bullet_{A_3(B_4\{(C_2,C_3)\},B_5)} A_3(B_4\{(\lambda, C_3)\}, B_5), \\ &\quad A_4\{(\lambda, B_7\{C_4\{D\}\})\} \bullet_{A_4\{B_6,B_7\{C_4\{D\}\}\}} A_4\{(B_6, \lambda)\}) \\ &= A(A_1(\lambda \bullet_{B_1} B_1, B_2 \bullet_{B_2} B_2, B_3\{\lambda\} \bullet_{B_3\{C_3\}} \lambda), \lambda \bullet_{A_2} A_2, \\ &\quad A_3(B_4\{(C_2, \lambda)\} \bullet_{B_4\{(C_2,C_3)\}} B_4\{(\lambda, C_3)\}, B_5 \bullet_{B_5} B_5), \\ &\quad A_4\{\lambda \bullet_{B_6} B_6, B_7\{C_4\{D\}\} \bullet_{B_7\{C_4\{D\}\}} \lambda\}) \\ &= A(A_1(\lambda, B_2, \lambda), \lambda, A_3(B_4\{\lambda\}, B_5), A_4\{(\lambda, \lambda)\}) \end{aligned}$$

According to the equivalence relation $\equiv$ we identifier $Y \bullet_X Z$ with

$$Y \bullet_X Z = A(A_1(B_2), A_3(B_4\{\lambda\}, B_5), A_4\{\lambda\}).$$

□

Let us finally summarise the results of this section. We have indeed shown that meets and joins exist in the poset (SubAttr(X), $\leq$), i.e., (SubAttr(X),$\bowtie_X$,$\bullet_X$) is a lattice, where X is the bottom element and λ is the top element. This lattice is finite, so it is trivialy also complete.

Proposition 4.19. *(SubAttr(X), $\bowtie_X$, $\bullet_X$, X, λ) with the partial order $\leq$ is a lattice with bottom element X and top element λ.* □

4.3 Distributivity and Relative Pseudo-Complements

In this section we investigate the structure of the lattice (SubAttr(X),$\leq$, $\bowtie_X$, $\bullet_X$, X, λ). We will show that distributivity laws hold, i.e., $\bowtie_X$ is distributive over $\bullet_X$, and $\bullet_X$ is distributive over $\bowtie_X$. As (SubAttr(X), $\leq$, $\bowtie_X$, $\bullet_X$, X, λ) is a finite lattice, the distributivity is already sufficient to conclude that SubAttr(X) carries the structure of a Heyting algebra. Recall from Definition 4.8 that this means that the relative pseudo-complement $Y \Longrightarrow Z$ exists for all $Y, Z \in$ SubAttr(X). In particular, taking Z as the bottom element, the pseudo-complement of Y as defined in Definition 4.23 will exist.

For the lattice of subattributes (SubAttr(X), $\leq$, $\bowtie_X$, $\bullet_X$, X, λ) we will use the notation $Y_X^{\mathcal{C}}$ for the pseudo-complement of Y. We will see that in general $(Y_X^{\mathcal{C}})_X^{\mathcal{C}} \neq Y$ holds, and SubAttr(X) cannot carry the structure of a Boolean algebra — this is already clear from cardinality considerations. Analogously, for the relative pseudo-complement in (SubAttr(X),$\leq$, $\bowtie_X$, $\bullet_X$, X, λ) we will use the notation $Y \to_X Z$, so $Y \to_X X = Y_X^{\mathcal{C}}$.

For our purpose here, knowing the existence of the (relative) pseudo-complement will not be enough. We need to know the exact structure of $Y \to_X Z$ and $Y_X^{\mathcal{C}}$. Therefore, we will give explicit definitions first for pseudo-complements, then for relative pseudo-complements.

Then we have to verify of course that these definitions will give indeed the pseudo-complement and the relative pseudo-complement, respectively.

4.3.1 Distributivity

We want to address the two distributivity laws in (SubAttr(X),$\leq$, $\bowtie_X$, $\bullet_X$, X, λ). Due to the following lemma it is enough to show one of them.

Lemma 4.20. *Suppose (Q, $\bowtie$, $\bullet$) is a lattice and $X, Y, Z \in Q$. If $\bowtie_Q$ is distributive over $\bullet_Q$, then $\bullet_Q$ is also distributive over $\bowtie_Q$.*

Proof.

$$\begin{aligned}(X \bullet_Q Z) \bowtie_Q (Y \bullet_Q Z) &= (X \bowtie_Q (Y \bullet_Q Z)) \bullet_Q (Z \bowtie_Q (Y \bullet_Q Z)) \\ &= ((X \bowtie_Q Y) \bullet_Q (X \bowtie_Q Z)) \bullet_Q ((Z \bowtie_Q Y) \bullet_X (Z \bowtie_Q Z)) \\ &= (X \bowtie_Q Y) \bullet_Q (X \bowtie_Q Z) \bullet_Q (Z \bowtie_Q Y) \bullet_Q Z \\ &= (X \bowtie_Q Y) \bullet_Q Z\end{aligned}$$

□

We verify the distributivity of (SubAttr(X), $\leq$, $\bullet_X$, $\bowtie_X$, X ,λ).

Proposition 4.21. *Let $Q \in \mathcal{NA}$ with $X, Y, Z \in$ Subattr(Q). Then $(X \bullet_Q Y) \bowtie_Q Z = (X \bowtie_Q Z) \bullet_Q (Y \bowtie_Q Z)$ and $(X \bowtie_Q Y) \bullet_Q Z = (X \bullet_Q Z) \bowtie_Q (Y \bullet_Q Z)$ hold.* □

Proof. We verify the distributive law $(X \bullet_Q Y) \bowtie_Q Z = (X \bowtie_Q Z) \bullet_Q (Y \bowtie_Q Z)$. If one of X or Y is λ, we immediately obtain $Z = Z$. For $Z = \lambda$ we obtain $X \bullet_Q Y = X \bullet_Q Y$. Thus, we may assume that X, Y and Z are all different from λ.

1. Assume $Q = \lambda$ and $Q \leq X, Y, Z$, i.e., $X = Y = Z = \lambda$. This case is already done.
2. Suppose Q is a flat attribute, say $Q = A$. Then, we have $X = Y = Z = A$, which implies that both sides evaluate to A.
3. Let Q be a tuple attribute, say $Q = A(A_1, \ldots, A_n)$, $1 \leq i \leq n$. For $Q \leq X, Y, Z$ we can take $X = A(A_1', \ldots, A_n')$, $Y = A(A_1'', \ldots, A_n'')$ and $Z = A(A_1''', \ldots, A_n''')$ with $A_i \leq A_i', A_i'', A_i'''$ for all $i = 1, \ldots n$.
   Then we get

$$\begin{aligned}&(X \bullet_Q Y) \bowtie_Q Z \\ &= A\Big((A_1' \bullet_{A_1} A_1''), \ldots, (A_n' \bullet_{A_n} A_n'')\Big) \bowtie_Q A(A_1''', \ldots, A_n''')\Big) \\ &= A\Big((A_1' \bullet_{A_1} A_1'') \bowtie_{A_1} A_1''', \ldots, (A_n' \bullet_{A_n} A_n'') \bowtie_{A_n} A_n'''\Big) \\ &= A\Big((A_1' \bowtie_{A_1} A_1''') \bullet_{A_1} (A_1'' \bowtie_{A_1} A_1'''), \ldots, (A_n' \bowtie_{A_n} A_n''') \bullet_{A_n} (A_n'' \bowtie_{A_n} A_n''')\Big) \\ &= A\Big(A_1' \bowtie_{A_1} A_1''', \ldots, A_n' \bowtie_{A_n} A_n'''\Big) \bullet_Q A\Big(A_1' \bowtie_{A_1} A_1''', \ldots, A_n'' \bowtie_{A_n} A_n'''\Big) \\ &= (X \bowtie_Q Z) \bullet_Q (Y \bowtie_Q Z).\end{aligned}$$

4. Finally, let $Q = A\{D\}$ and $Q \leq X, Y, Z$. So we may assume $X = A\{A'\}$, $Y = A\{B\}$ and $Z = A\{C\}$ with $D \leq A, B, C$. Then the following equation holds:

$$\begin{aligned}(X \bullet_Q Y) \bowtie_Q Z &= A\{(A' \bullet_D B)\} \bowtie_Q A\{C\} = A\{(A' \bullet_D B) \bowtie_D C\} \\ &= A\{(A' \bowtie_D C) \bullet_D (B \bowtie_D C)\} \\ &= A\{(A' \bowtie_D C)\} \bullet_{A\{D\}} A\{(B \bowtie_D C)\} \\ &= (X \bowtie_Q Z) \bullet_Q (Y \bowtie_Q Z).\end{aligned}$$

This completes the proof.

□

According to [EoI00, p. 125] the following proposition holds.

Proposition 4.22. *Any finite distributive lattice is a Heyting lattice.* □

This proof is obvious, as we only have to show the existence of relative pseudo-complements. As these are defined by suprema, the existence is obvious in a finite lattice.

So (SubAttr(X),$\leq$,$\bowtie_X$,$\bullet_X$, X, λ) is a Heyting algebra. However, we will not obtain the structure of a Boolean Algebra as in case of the flat attributes. In the next section we consider the complement operation $Y_X^{\mathcal{C}}$ on each element of SubAttr(X). We discover that due to the fact that $(Y_X^{\mathcal{C}})_X^{\mathcal{C}} \leq Y$ does not hold in general, we do not have a Boolean algebra.

4.3.2 The Pseudo-Complement and the Relative Pseudo-Complement

In this section we consider the structure of the complement operation on SubAttr(X). The pseudo-complement $Y_X^{\mathcal{C}}$ can be defined as follows.

Definition 4.23 (*pseudo-complement*). Let X, Y be nested attributes with $X \leq Y$. Then we define the pseudo-complement of Y with respect to X, denoted by $Y_X^{\mathcal{C}}$, as follows:

1. Suppose $Y = \lambda$. Then
$$(\lambda)_X^{\mathcal{C}} := X.$$
2. If $Y = X$, then
$$(Y)_X^{\mathcal{C}} = \lambda.$$
3. If $Y = A(A_1', \ldots, A_n')$ is a subattribute of $X = A(A_1, \ldots, A_n)$ with $A_j \leq A_j'$ for all $1 \leq j \leq n$, then
$$(Y)_X^{\mathcal{C}} := A((A_1')_{A_1}^{\mathcal{C}}, \ldots, (A_n')_{A_n}^{\mathcal{C}})$$
with $(A_i')_{A_i}^{\mathcal{C}}$ and $A_i < A_i'$ for all $1 \leq i \leq n$.

4. If $Y = A\{B\}$ is a subattribute of $X = A\{D\}$ with $D \leq B$, then

$$(Y)^{\mathcal{C}}_X = (A\{B\})^{\mathcal{C}}_{A\{D\}} := A\{(B)^{\mathcal{C}}_D\}.$$

□

We have to show that $Y^{\mathcal{C}}_X$ really defines the pseudo-complement of Y. This will be done in Proposition 4.27. Before we do this, we first give an example for the pseudo-complement $Y^{\mathcal{C}}_X$.

Example 4.24. Again we use the same subattributes as in Example 4.15, i.e.,

$$\begin{aligned}
X &= A(A_1(B_1, B_2, B_3\{C_1\}), A_2, A_3(B_4\{(C_2, C_3)\}, B_5), A_4\{(B_6, B_7\{C_4\{D\}\})\}) \\
Y &= A(A_1(B_2, B_3\{\lambda\}), A_3(B_4\{C_2\}, B_5), A_4\{(B_7\{C_4\{D\}\})\}) \\
Z &= A(A_1(B_1, B_2), A_2, A_3(B_4\{C_3\}, B_5), A_4\{(B_6)\})
\end{aligned}$$

Again we identify Y with

$$Y = A(A_1(\lambda, B_2, B_3\{\lambda\}), \lambda, A_3(B_4\{(C_2, \lambda)\}, B_5), A_4\{(\lambda, B_7\{C_4\{D\}\})\}))$$

using the equivalence relation $\equiv$ on $\mathrm{SubAttr}(X)$. Analogously, we identify Z with

$$Z = A(A_1(B_1, B_2, \lambda), A_2, A_3(B_4\{(\lambda, C_3)\}, B_5), A_4\{(B_6, \lambda)\}).$$

For the pseudo-complement $Y^{\mathcal{C}}_X$ we get:

$$\begin{aligned}
Y^{\mathcal{C}}_X &= A(A_1(\lambda, B_2, B_3\{\lambda\})^{\mathcal{C}}_{A_1(B_1,B_2,B_3\{C_1\})}, (\lambda)^{\mathcal{C}}_{A_2}, \\
&\qquad A_3(B_4\{(C_2, \lambda)\}, B_5)^{\mathcal{C}}_{A_3(B_4\{(C_2,C_3)\},B_5)}, A_4\{(\lambda, B_7\{(C_4\{D\})\})\}) \\
&= A(A_1((\lambda)^{\mathcal{C}}_{B_1}, (B_2)^{\mathcal{C}}_{B_2}, (B_3\{\lambda\})^{\mathcal{C}}_{B_3\{C_1\}}), (\lambda)^{\mathcal{C}}_{A_2}, \\
&\qquad A_3((B_4\{(C_2, \lambda)\})^{\mathcal{C}}_{B_4\{(C_2,C_3)\}}, (B_5)^{\mathcal{C}}_{B_5}), A_4\{(\lambda)\}^{\mathcal{C}}_{B_6}, (B_7\{C_4\{D\}\})^{\mathcal{C}}_{B_7\{C_4\{D\}\}}\}) \\
&= A(A_1((\lambda)^{\mathcal{C}}_{B_1}, (B_2)^{\mathcal{C}}_{B_2}, (B_3\{(\lambda)^{\mathcal{C}}_{C_1}\})), (\lambda)^{\mathcal{C}}_{A_2}, \\
&\qquad A_3(B_4\{((C_2)^{\mathcal{C}}_{C_2}, (\lambda)^{\mathcal{C}}_{C_3})\}), (B_5)^{\mathcal{C}}_{B_5}), A_4\{((\lambda)^{\mathcal{C}}_{B_6}, B_7\{(C_4\{D\})^{\mathcal{C}}_{C_4\{D\}}\})\}) \\
&= A(A_1(B_1, \lambda, B_3\{C_1\}), A_2, A_3(B_4\{(\lambda, C_3)\}, \lambda), A_4\{(B_6, B_7\{\lambda\})\})
\end{aligned}$$

Using the equivalence relation $\equiv$ we obtain

$$Y^{\mathcal{C}}_C = A(A_1(B_1, B_3\{C_1\}), A_2, A_3(B_4\{(C_3)\}), A_4\{(B_6, B_7\{\lambda\})\})$$

as pseudo-complement of Y.

The pseudo-complement $Y^{\mathcal{C}}_X$ is not a complement, as the following example demonstrates.

Example 4.25. Assume $X = A\{B(C_1, C_2)\}$ and $Y = A\{B(C_1)\}$. Then we have $B(C_1, C_2) \leq B(C_1)$ and $C_1 \neq C_2$. Then

$$(Y)^{\mathcal{C}}_X = (A\{B(C_1)\})^{\mathcal{C}}_{A\{B(C_1,C_2)\}} = A\{B(C_1)^{\mathcal{C}}_{B(C_1,C_2)}\} = A\{B(C_2)\}.$$

If $(Y)^{\mathcal{C}}_X$ were a complement, then

$$Y \bowtie_X Y^{\mathcal{C}} = X \text{ and } Y \bullet_X Y^{\mathcal{C}} = \lambda$$

would hold. However, we find that

$$Y \bowtie_X Y^{\mathcal{C}} = A\{B(C_1)\} \bowtie_{A\{B(C_1,C_2)\}} A\{B(C_2)\} = X,$$

but

$$Y \bullet_X Y^{\mathcal{C}} = A\{B(C_1)\} \bullet_{A\{B\{(C_1,C_2)\}} A\{B(C_2)\} = A\{B(C_1) \bullet_{B(C_1,C_2)} B(C_2)\} = A\{\lambda\} \neq \lambda.$$

This means that $(\mathrm{SubAttr}(X), \leq, \bowtie_X, \bullet_X, X, \lambda)$ is not a Boolean algebra. □

We summarise the last observation in a proposition.

Proposition 4.26. *$(SubAttr(X), \leq, \bowtie_X, \bullet_X, X, \lambda)$ is not a Boolean algebra.* □

Up to now we have only defined an operation on $\mathrm{Subattr}(X)$ and called it a pseudo-complement. However, we still have to show that $Y^{\mathcal{C}}_X$ is really the pseudo-complement of Y in $\mathrm{SubAttr}(X)$. This will be done in the next propostion.

Proposition 4.27. *$Y^{\mathcal{C}}_X$ is the pseudo-complement of Y in $SubAttr(X)$.* □

Proof. If $Y^{\mathcal{C}}_X$ ist the pseudo-complement of Y in $\mathrm{SubAttr}(X)$, then we have to show that the following equation holds.

$$Y^{\mathcal{C}}_X = \bullet_X\{X' \in \mathrm{SubAttr}(X) \mid X' \bowtie_X Y \leq X\}.$$

Therefore we consider the different cases in which Y can occur.

1. If $Y = \lambda$. Then only $X' = X$ satisfies the equation $X' \bowtie_X \lambda \leq X$, i.e., $\bullet_X\{X' \in \mathrm{SubAttr}(X) \mid X' \bowtie_X Y \leq X\} = X$ in accordance with Definition 4.23.

2. For $Y = X$ all $X' \in$Subattr(X) satisfy $X' \bowtie_X Y \leq X$. This implies $\bullet_X\{X' \in \mathrm{SubAttr}(X) \mid X' \bowtie_X Y \leq X\} = \lambda$ in accordance with Definition 4.23.

3. If X is a tuple attribute, say $X = A(A_1, \ldots, A_n)$, we can take $Y = A(A'_1, \ldots, A'_n)$ with $A_i \leq A'_i$ for all $1 \leq i \leq n$. By induction we obtain

   $$(A'_i)^{\mathcal{C}}_{A_i} = \bullet_{A_i}\{X' \in \mathrm{SubAttr}(A_i) \mid X' \bowtie_{A_i} A'_i \leq A_i\}$$

   holds for all A'_i with $1 \leq i \leq n$. This implies that the X' with $X' \bowtie_X Y \leq X$ have the form $A(X'_1, \ldots, X'_n)$ with $X'_i \bowtie_{A_i} A'_i \leq A_i$. This implies $\bullet_X\{X' \in \mathrm{SubAttr}(X) \mid X' \bowtie_X Y \leq X\} = A((A'_1)^{\mathcal{C}}_{A_i}, \ldots, (A'_n)^{\mathcal{C}}_{A_n})$ in accordance to Defintion 4.23.

4. If $Y = A\{B\}$ is subattribute of $X = A\{D\}$ with $B, D \in \mathcal{NA}$ and $D \leq B$, then by induction $(B)^{\mathcal{C}}_D = \bullet_D\{X' \in \text{Subattr}(D) \mid X' \bowtie_D B \leq D\}$ holds. As the X' with $X' \bowtie_X Y \leq x$ all have the form $A\{X''\}$ with $X'' \bowtie_D B \leq D$, we conclude $\bullet_X\{X' \mid X' \bowtie_X Y \leq X\} = A\{(B)^{\mathcal{C}}_D\} = (A\{B\})^{\mathcal{C}}_{A\{D\}}$ in accordance with Definition 4.23.

□

Analogous to Definition 4.23 we now define the pseudo-complement of Y relative to Z in X.

Definition 4.28 (relative pseudo-complement). In the lattice $(\text{SubAttr}(X), \bowtie_X, \bullet_X)$ the *pseudo-complement of Y relative to Z*, denoted as $Y \to_X Z$, is defined as follows:

1. If $Y \leq Z$ holds, then $Y \to_X Z = \lambda$.
2. For any subattribute Z of X, i.e., $X \leq Z$ we define $\lambda \to_X Z = Z$.
3. Assume $X = A(A_1, \ldots, A_n)$ is a tuple attribute. Then we can take $Y = A(A_1', \ldots, A_n')$ and $Z = A(A_1'', \ldots, A_n'')$ with $A_j \leq A_j', A_j''$ for all $j = 1, \ldots, n$, but $A_j' \not\leq A_j''$ for at least one index j. In this case we define
$$Y \to_X Z := A(A_1' \to_{A_1} A_1'', \ldots, A_n' \to_{A_n} A_n'').$$
4. Assume $X = A\{D\}$ is a set attribute and $X \leq Y, Z$ with $Y \not\leq Z$. We may assume $Y = A\{C\}$ and $Z = A\{B\}$ with $D \leq B, C$ and $C \not\leq B$. Then we define
$$Y \to_X Z := A\{C \to_D B\}.$$

Again, we have to show that Definition 4.28 gives really the relative pseudo-complement of Y relative to Z. This will be shown in Proposition 4.30. Before we do this, we present an example for the relative pseudo-complement $Y \to_X Z$.

Example 4.29. We use the same subattributes as in Example 4.15, i.e.,

$$X = A(A_1(B_1, B_2, B_3\{C_1\}), A_2, A_3(B_4\{(C_2, C_3)\}, B_5), A_4\{(B_6, B_7\{C_4\{D\}\})\})$$
$$Y = A(A_1(B_2, B_3\{\lambda\}), A_3(B_4\{C_2\}, B_5), A_4\{(B_7\{C_4\{D\}\})\})$$
$$Z = A(A_1(B_1, B_2), A_2, A_3(B_4\{C_3\}, B_5), A_4\{(B_6)\})$$

We identify Y with

$$Y = A(A_1(\lambda, B_2, B_3\{\lambda\}), \lambda, A_3(B_4\{(C_2, \lambda)\}, B_5), A_4\{(\lambda, B_7\{C_4\{D\}\})\}))$$

using the equivalence relation $\equiv$ on $\text{SubAttr}(X)$. Analogously, we identify Z with

$$Z = A(A_1(B_1, B_2, \lambda), A_2, A_3(B_4\{(\lambda, C_3)\}, B_5), A_4\{(B_6, \lambda)\}).$$

For the the pseudo-complement $Y \to_X Z$ we get:

$$\begin{aligned}
Y \to_X Z &= A(A_1(\lambda, B_2, B_3\{\lambda\}) \to_{A_1(B_1,B_2,B_3\{C_1\})} A_1(B_1, B_2, \lambda), \lambda \to_{A_2} A_2 \\
&\quad A_3(B_4\{(C_2, \lambda)\}, B_5) \to_{A_3(B_4\{(C_2,C_3)\},B_5)} A_3(B_4\{(\lambda, C_3)\}, B_5), \\
&\quad A_4\{(\lambda, B_7\{C_4\{D\}\})\} \to_{A_4\{(B_6,B_7\{C_4\{D\}\})\}} A_4\{(B_6, \lambda)\}) \\
&= A(A_1(\lambda \to_{B_1} B_1, B_2 \to_{B_2} B_2, B_3\{\lambda\} \to_{B_3\{C_1\}} \lambda), \lambda \to_{A_2} A_2, \\
&\quad A_3(B_4\{(C_2, \lambda)\} \to_{B_4\{(C_2,C_3)\}} B_4\{(\lambda, C_3)\}, B_5 \to_{B_5} B_5), \\
&\quad A_4\{(\lambda \to_{B_6} B_6, B_7\{D\} \to_{B_7\{D\}} \lambda)\}) \\
&= A(A_1(\lambda \to_{B_1} B_1, B_2 \to_{B_2} B_2, B_3\{\lambda\} \to_{B_3\{C_1\}} \lambda), \lambda \to_{A_2} A_2, \\
&\quad A_3(B_4\{(C_2 \to_{C_2} \lambda, \lambda \to_{C_3} C_3)\}, B_5 \to_{B_5} B_5), \\
&\quad A_4\{(\lambda \to_{B_6} B_6, B_7\{D\} \to_{B_7\{D\}} \lambda)\}) \\
&= A(A_1(B_1, \lambda, \lambda), A_2, A_3(B_4\{(\lambda, C_3)\}, \lambda), A_4\{(B_6, \lambda)\})
\end{aligned}$$

Using the equivalence relation $\equiv$, we can identify $Y \to_X$ with

$$Y \to_X Z = A(A_1(B_1), A_2, A_3(\{B_4\{C_3\}), \lambda).$$

In order to make sure that the definition above really defines the pseudo-complement of Y relative to Z in $\mathrm{SubAttr}(X)$ we have to show, that

$$Y \to_X Z = \bullet_X\{X' \in \mathrm{SubAttr}(X) \mid X' \bowtie_X Y \leq Z\}$$

holds. Equivalently — and this seems to be easier — we can show that for all subattributes $X' \in \mathrm{SubAttr}(X)$

$$X' \leq (Y \to_X Z) \iff X' \bowtie_X Y \leq Z$$

holds. We verify this equivalence in the following Lemma 4.30. From this we conclude that $Y \to_X Z = \bullet_X\{X' \in \mathrm{SubAttr}(X) \mid X' \bowtie_X Y \leq Z\}$ holds.

Lemma 4.30. *For all $X', Y, Z \in$ SubAttr(X) the equivalence*

$$X' \leq (Y \to_X Z) \iff X' \bowtie_X Y \leq Z$$

holds. □

Proof. "$\Longrightarrow$": We first show that $X' \leq (Y \to_X Z)$ implies $X' \bowtie_X Y \leq Z$.

If $X' \leq (Y \to_X Z)$ holds $Y \bowtie_X X' \leq Y \bowtie_X (Y \to_X Z)$ follows, and it remains to show that $Y \bowtie_X (Y \to_X Z) \leq Z$. For $Y = \lambda$ we get $Y \to_X Z = Z$ and then, $Y \bowtie_X (Y \to_X Z) = Y \bowtie_X Z \leq Z$. For $Y \leq Z$ we get $Y \to_X Z = \lambda$ and thus, $Y \bowtie_X (Y \to_X Z) = Y \leq Z$. So we may assume $Y \neq \lambda$ and $Y \not\leq Z$. We proceed by induction on X. For $X = \lambda$ and X being a flat attribute there is nothing left to show.

1. Let X be a tuple attribute, say $X = A(A_1, \ldots, A_n)$. Then we may assume $Y = A(A_1', \ldots, A_n')$ and $Z = A(A_1'', \ldots, A_n'')$ with $A_i \leq A_i', A_i''$ for all $1 \leq i \leq n$. Furthermore, for at least one i we have $A_i' \not\leq A_i''$. Then we obtain

$$\begin{aligned} Y \bowtie_X (Y \to_X Z) &= A(A_1', \ldots, A_n') \bowtie_{A(A_1,\ldots,A_n)} \left(A(A_1' \to_{A_1} A_1'', \ldots, A_n' \to_{A_n} A_n'')\right) \\ &= A(A_1' \bowtie_{A_1} (A_1' \to_{A_1} A_1''), \ldots, A_n' \bowtie_{A_n} (A_n' \to_{A_n} A_n'')). \end{aligned}$$

   By induction we have $A_i' \bowtie_{A_i} (A_i' \to_{A_i} A_i'') \leq A_i''$ for all $i = 1, \ldots, n$, which implies

$$Y \bowtie_X (Y \to_X Z) \leq A(A_1'', \ldots, A_n'') = Z.$$

2. Let $X = A\{D\}$ be a set attribute with $X \leq Y, Z$ and $Y \not\leq Z$. We may assume $Y = A\{C\}$, $Z = A\{B\}$ with $D \leq B, C$ and $C \not\leq B$. By induction $C \bowtie_D (C \to_D B) \leq B$. This implies

$$Y \bowtie_X (Y \to_X Z) = A\{C\} \bowtie_{A\{D\}} A\{C \to_D B\} = A\{C \bowtie_D (C \to_D B)\} \leq A\{B\} = Z,$$

   which completes the proof.

"$\Longleftarrow$": Conversely, we have to show that $X' \bowtie_X Y \leq Z$ implies $X' \leq (Y \to_X Z)$. For $Y = \lambda$ we have $Y \to_X Z = Z$ and $X' \bowtie_X Y = X'$. So both sides of the implication are $X' \leq Z$ and nothing is to show. For $Y \leq Z$ we have $Y \to_X Z = \lambda$. So the right hand side of the implication is $X' \leq \lambda$, which holds anyway. For $X' = \lambda$ the left hand side of the implication is $Y \leq Z$, which implies $X' = \lambda \leq \lambda = (Y \to_X Z)$.

We show the remaining cases by induction on X. There is nothing to show for $X = \lambda$ or X being a flat attribute.

1. Let X be a tuple attribute, $X = A(A_1, \ldots, A_n)$, and $X \leq X', Y, Z$ with $Y \neq \lambda$, $Z \neq \lambda$ and $X' \neq \lambda$. In this case we can take $Y = A(A_1', \ldots, A_n')$, $Z = A(A_1'', \ldots, A_n'')$ and $X' = A(A_1''', \ldots, A_n''')$. There is at least one i with $A_i' \not\leq A_i''$. The left hand side of the implication becomes $A(A_1''' \bowtie_{A_1} A_1', \ldots, A_n''' \bowtie_{A_n} A_n') \leq A(A_1'', \ldots, A_n'')$, which implies with $(A_i''' \bowtie_{A_i} A_i') \leq A_i''$ for all $1 \leq i \leq n$. By induction this further implies $A_i''' \leq (A_i' \to_{A_i} A_i'')$ for all $i = 1, \ldots, n$. From this we conclude $A(A_1''', \ldots, A_n''') \leq A(A_1' \to_{A_1} A_1'', \ldots, A_n' \to_{A_n} A_n'')$, which is the right hand side of the implication.

2. Let $X = A\{D\}$ and $X \leq Y, Z, X'$ with $Y \neq \lambda$, $X' \neq \lambda$ and $Y \neq Z$. We can take $Y = A\{C\}$, $Z = A\{B\}$ with $C \not\leq B$ and $X' = A\{E\}$, $D \leq B, C, E$. Then the left hand side of the implication becomes $A\{E \bowtie_D C\} \leq A\{B\}$, which implies $E \bowtie_D C \leq B$. By induction we obtain $E \leq (C \to_D B)$ and thus $A\{E\} \leq A\{C \to_D B\}$, which is the right hand side of the implication.

□

The next proposition is an immediate consequence of Lemma 4.30.

Proposition 4.31. *$Y \rightarrow_X Z$ is the pseudo-complement of Y relative to Z in SubAttr(X), i.e., the equation*

$$Y \rightarrow_X Z = \bullet_X\{X' \in SubAttr(X) \mid X' \bowtie_X Y \leq Z\}$$

holds. □

Corollary 4.32. *The pseudo-complement of Y relative to Z in X with $Z = X$ is the pseudo-complement of Y relative to X as defined in Definition 4.23.*

Proof. We just replace Z by X in the Proposition 4.31 and use Proposition 4.27. □

Thus carries the structure of a Heyting Algebra with the join $\bullet_X$, the meet $\bowtie_X$, pseudo-complement $(.)^c_X$ and relative pseudo-complement $\rightarrow_X$.

Chapter 5

Functional Dependencies in HERM

This chapter contains the central result of this thesis. In Chapter 4 we studied the structure of the set of subattributes of a given attribute and dicovered that they carry the structure of a Heyting algebra. We now generalise the axiomatisation of functional dependencies from the Relational Database Model to the Higher-Order Entity-Relationship Model (HERM).

According to Definition 2.16 axiomatisability means to define a finite set of rules and axioms and to verify its soundness and completeness. Therefore, we approach the problem in two steps. In the first step, we present a system of rules and axioms, which generalises the Armstrong-axioms known from the RDM. For this set of axioms and rules we then show that it is sound, i.e., for each given set dep of functional dependencies we obtain $dep^+ \subseteq dep^*$. This means that every syntactically derivable functional dependency is also semantically implied. The proof will simply investigate the models of functional dependencies. In the second step we show that any semantically implied functional dependency is syntactically derivable, i.e., that $dep^* \subseteq dep^+$ holds. The proof will start with a non-derivable functional dependency ϕ and construct a model for dep^*, which is not a model for ϕ.

Finally, it is easy to see that the generalised Armstrong-axioms are a minimal system with these properties. In particular, they include the implication of functional dependencies in the RDM as a simple subcase.

5.1 Functional Dependencies in HERM

In this chapter we consider functional dependencies in HERM based on the corresponding nested attribute N_R. We start with the generalisation of the definitions, which we used for functional dependencies in the RDM. Analogously to Definition 2.9 we introduce functional dependencies in the HERM.

Definition 5.1 (Functional Dependencies, Satisfaction). Let S be a HERM schema. A *functional dependency* on S is an expression of the from $X \to Y$ with $X, Y \in \text{SubAttr}(N_R)$. A set $R^t \subseteq Dom(N_R)$ *satisfies* a functional dependency $X \to Y$ if and only if for all t_1, t_2 $\pi_X^{N_R}(t_1) = \pi_X^{N_R}(t_2)$ implies $\pi_Y^{N_R}(t_1) = \pi_Y^{N_R}(t_2)$. In this case we call R^t a *model* for $X \to Y$.

□

In the following we simply talk of relationships and dispense with entities as they are just a special type of relationships. An entity type is a relationship type R of order 0, i.e., with an empty set $comp(R)$ of components.

As in case of the RDM, we ask how to determine the set of implied functional dependencies. Again, we define the implication and the semantic hull.

Definition 5.2 (implication, semantic hull). Let ϕ and ψ be functional dependencies on a HERM schema S.

We say that ψ is an *implication* of ϕ ($\phi \models \psi$ for short) if and only if each model for ϕ is also a model for ψ.

ψ is an *implication* of a set *dep* of dependencies, if and only if each $R^t \subseteq Dom(N_R)$ that is a common model of *dep*, i.e., is a model of all $\phi \in dep$, is also a model of ψ ($dep \models \psi$ for short).

The *semantic hull* dep^* of a set *dep* of dependencies is defined as the set of all implied dependencies of *dep* ($dep^* = \{\psi \mid dep \models \psi\}$). □

Next, we define derivation rules and axioms. Again, we choose parameterised functional dependencies in the HERM. Similar to the RDM we have functional dependencies in the HERM, in which variables may occur. The choosen variables can present flat, tuple or set attributes, but we can restrict our attention to subattributes on N_R. In these cases, the only boundary conditions that can occur have the form $Y \leq X$.

Definition 5.3 (derivation rule, premises, conclusion, axiom). *A* derivation rule for functional dependencies in the HERM consists of a finite set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of parameterised functional dependencies, a conclusion $\mathcal{C}$, which also is a parameterised functional dependency, and a finite set $Cond = \{C_1, \ldots, C_k\}$ of conditions on the parameters in $\mathcal{P}$ and $\mathcal{C}$. The functional dependencies P_i $(i = 1, \ldots, n)$ are called the *premises* of the rule and the functional dependency $\mathcal{C}$ is called *conclusion* of the rule. A derivation rule without premises ($\mathcal{P} = \emptyset$) is called an *axiom*. □

Given a set *dep* of functional dependencies on a HERM schema S, we again regard these dependencies simply as purely syntactical expressions. Derivation rules then tell us which additional expressions can be derived. The notation remains to be the same as in case of the RDM.

Then the derivation tree is defined as in Definition 2.13. It represents a finite sequence of applications of rules starting with same given set of functional dependencies.

We now have to introduce the syntactic hull of a set of functional dependencies in the HERM. Again, we define it in the same way as in the RDM.

Definition 5.4 (derivable, syntactic hull). Le $\mathcal{R}$ be a set of axioms and rules and let *dep* be a set of functional dependencies in a HERM schema S. A functional dependency is *derivable* from *dep* using $\mathcal{R}$ if and only if there exists a derivation tree over $\mathcal{R}$ and *dep* with root ψ (notation $dep \vdash_{\mathcal{R}} \psi$).

The *syntactic hull* dep^+ of *dep* under $\mathcal{R}$ is the set of all functional dependencies that are derivable from *dep* using $\mathcal{R}$, i.e., $dep^+ = \{\psi \mid \vdash_{\mathcal{R}} \psi\}$. □

We are no ready to prove of the axiomatisability of functional dependencies in the HERM. We start proving the soundness of the Armstrong-axioms.

5.2 Soundness of the Armstrong-axioms

We now define the generalised Armstrong-axioms. The Armstrong-axioms for the RDM that are defined in Definition 2.17 provide a special case of this definition.

Definition 5.5 (generalised Armstrong-axioms). The *generalised Armstrong-axioms* for the implication of functional dependencies are:

1. **reflexivity axiom**:
$$\frac{}{X \to Y} X \leq Y$$
2. **augmentation rule:**
$$\frac{X \to Y}{X \to X \bowtie_{N_R} Y}$$
3. **transitivity rule:**
$$\frac{X \to Y, Y \to Z}{X \to Z}$$

□

Proposition 5.6. *The generalised Armstrong-axioms are sound.* □

Proof. Assume r is a relationship set over R. We have to show that whenever r is a model for the premises of a rule in Definition 5.5, it is also a model for the conclusion.

Reflexivitiy axiom:
Assume $t_1|_X = t_2|_X$ and $X \leq Y$. Using the mapping $\pi^X_Y : \text{Dom}(X) \to \text{Dom}(Y)$ introduced

in Definition 3.9, we have $t_1|_X = \pi_X^{N_R}(t_1)$ and $\pi_Z^X \circ \pi_Y^X = \pi_Z^X$ we obtain the following implication:

$$\begin{aligned} t_1 \mid_X = t_2 \mid_X &\Longrightarrow \pi_Y^X(\pi_X^{N_R}(t_1)) = \pi_Y^X(\pi_X^{N_R}(t_2)) \\ &\Longrightarrow \pi_Y^{N_R}(t_1) = \pi_Y^{N_R}(t_2) \\ &\Longrightarrow t_1 \mid_Y = t_2 \mid_Y . \end{aligned}$$

Augmentation rule:
Assume that $\models_r$: $X \to Y$ holds. We have to show $\models_r$: $X \to X \bowtie_{N_R} Y$. So take $t_1, t_2 \in r$ with $t_1 \mid_X = t_2 \mid_X$. The assumption $\models_r$: $X \to Y$ implies also $t_1 \mid_Y = t_2 \mid_Y$. Thus, we have to show $t_1 \mid_{X \bowtie_{N_R} Y} = t_2 \mid_{X \bowtie_{N_R} Y}$. However, we will show $t_1 \mid_A = t_2 \mid_A$ for all subattributes A of $X \bowtie_{N_R} Y$.

1. For $A = \lambda$ there is nothing to show, as $t_1 \mid_\lambda = t_2 \mid_\lambda$ always holds.

2. Assume A is a flat attribute. Then we have $X \leq A$ or $Y \leq A$. As we assume $t_1 \mid_X = t_2 \mid_X$ and $t_1 \mid_Y = t_2 \mid_Y$, the claim $t_1 \mid_A = t_2 \mid_A$ follows immediately.

3. Assume A is a tuple attribute. In this case we may assume $A = R(A_1''', \ldots, A_n''')$ for $N_R = R(A_1, \ldots, A_n)$ such that $A_i \leq A_i'''$ holds for all $1 \leq i \leq n$. Furthermore, we have $X = R(A_1', \ldots, A_n')$ and $Y = R(A_1'', \ldots, A_n'')$ with $A_i' \bowtie_{A_i} A_i'' \leq A_i'''$ for all $i = 1, \ldots, n$. The assumption $t_1 \mid_X = t_2 \mid_X$ and $t_1 \mid_Y = t_2 \mid_Y$ implies $t_1 \mid_{R(A_i')} = t_2 \mid_{R(A_i')}$ and $t_1 \mid_{R(A_i'')} = t_2 \mid_{R(A_i'')}$ for all $i = 1, \ldots, n$. By induction we obtain $t_1 \mid_{R(A_i''')} = t_2 \mid_{R(A_i''')}$ for all $i = 1, \ldots, n$, which implies $t_1 \mid_A = t_2 \mid_A$.

4. Assume $A = R\{B\}$ is a set attribute for $N_R = R\{Z\}$ such that $Z \leq B$ holds. Then we also have $X = A\{D\}$ and $Y = A\{C\}$ wtith $Z \leq D, C$. We know $D \bowtie C \leq B$, which by induction implies $t_1 \mid_{R\{B\}} = t_2 \mid_{R\{B\}}$ as derived.

Transitivity rule:
Assume that $\models_r$: $X \to Y$ and $\models_r$: $Y \to Z$ hold. We have to show $\models_r$: $X \to Z$. So let $t_1, t_2 \in r$ with $t_1 \mid_X = t_2 \mid_X$. As we assume $\models_r$: $X \to Y$, we obtain $t_1 \mid_Y = t_2 \mid_Y$. From this we infer $t_1 \mid_Z = t_2 \mid_Z$, because we have $\models_r$: $Y \to Z$. This completes the proof. □

Before we approach the completeness of the Armstrong-axioms we show how to use them to derive further derivation rules.

Proposition 5.7. *The following derivation rules can be derived from the Armstrong-*

axioms:

$$\frac{X \to Y, X \to Z}{X \to X \bowtie_{N_R} Z} \qquad \textit{(meet rule)}$$

$$\frac{X \to Y, X \to Z}{X \to X \bullet_{N_R} Z} \qquad \textit{(join rule)}$$

$$\frac{X \to Y, X \to Z}{X \to (Y \to_{N_R} Z)} \qquad \textit{(pseudo-complement rule)}$$

$$\frac{X \to Y}{X \to A} Y \leq A \qquad \textit{(fragmentation rule)}$$

$$\frac{X \to Y}{U \to V} U \leq X, X \bowtie_{N_R} Y \leq V \qquad \textit{(general extension rule)}$$

$$\frac{X \to Y, U \to V}{W \to Z} X \bowtie_{N_R} Y \leq U, W \leq X, V \bowtie_{N_R} W \leq Z \qquad \textit{(general transitivity rule)}$$

□

Proof. We derive the rules.

meet rule:

$$\dfrac{\dfrac{X \to Y}{X \to X \bowtie_{N_R} Y} \quad \dfrac{\dfrac{\overline{X \bowtie_{N_R} Y \to X} \quad X \to Z}{X \bowtie_{N_R} Y \to Z}}{X \bowtie_{N_R} Y \to X \bowtie_{N_R} Y \bowtie_{N_R} Z} \quad \overline{X \bowtie_{N_R} Y \bowtie_{N_R} Z \to Y \bowtie_{N_R} Z}}{X \bowtie_{N_R} Y \to Y \bowtie_{N_R} Z}$$

$$X \to Y \bowtie_{N_R} Z$$

join rule:

$$\frac{X \to Y, \quad \overline{Y \to Y \bullet_{N_R} Z}}{X \to X \bullet_{N_R} Z}$$

pseudo-complement rule:

$$\frac{X \to Z \quad \overline{Z \to (Y \to_{N_R} Z)}}{X \to (Y \to_{N_R} Z)}$$

fragmentation rule:

$$\frac{X \to Y \quad \overline{Y \to A}}{X \to A}$$

general extension rule:

$$\frac{\dfrac{\overline{U \to X} \quad \dfrac{X \to Y}{\overline{X \to X \bowtie_{N_R} Y}}}{U \to X \bowtie_{N_R} Y} \qquad \overline{X \bowtie_{N_R} Y \to V}}{U \to V}$$

general transitivity rule:

$$\frac{\dfrac{\dfrac{\dfrac{\overline{W \to X} \quad \dfrac{X \to Y}{\overline{X \to X \bowtie_{N_R} Y}}}{W \to X \bowtie_{N_R} Y} \qquad \overline{X \bowtie_{N_R} Y \to U}}{W \to U} \qquad U \to V}{W \to V} \qquad \overline{V \bowtie_{N_R} W \to Z}}{\dfrac{W \to V \bowtie_{N_R} W}{W \to Z}}$$

□

5.3 Completeness of the Armstrong-axioms

In this section we show the completeness of the Armstrong-axioms. The proof follows the same idea as the completeness proof for the Armstrong-axioms in case of the RDM. In order to show $dep^* \subseteq dep^+$ we take a functional dependency ϕ not in dep^+ and construct a model for dep^*, which is not a model for ϕ. Similar to the case of the RDM (see e.g. [Sch97, p. 61]), the construction exploits the closure of a subattribute X, which we define next.

Definition 5.8 (closure). Let X be a subattribute of N_R. We define the *closure* $\overline{X}$ *of* X as

$$\overline{X} := \bowtie_{N_R} \{N_R \leq Z | X \to Z \in dep^+\}.$$

□

The closure $\overline{X}$ is well defined, because the set $\{N_R \leq Z | X \to Z \in Dep^+\}$ is finite. Thus, the meet $\bowtie_{N_R}$ is defined. In particular, we have $\bowtie_{N_R} \{Z\} = Z$ if Z for a single attribute Z and $\bowtie_{N_R} \emptyset = \lambda$.

We will need the following technical lemma in the proof of the main result.

Lemma 5.9. *Let X, Y, Z be subattributes of N_R with $X \leq Y, Z$ and $Z \not\leq Y$. Then there exists a subattribute $W \neq \lambda$ with $Y \leq W$ and $Z_X^{\mathcal{C}} \leq W$.* □

Proof. We use structural induction on N_R. Without loss of generality we may exclude $Z = \lambda$, as in this case $Z_X^{\mathcal{C}} = X$ would hold, which immediately allows us to chose $W = Y$.

1. Assume $N_R = \lambda$. This implies $X = Y = Z = \lambda$ and $Z \not\leq Y$ cannot be met.

2. Assume that N_R is a flat attribute, say $N_R = A$. As the only subattributes are A and λ, we must have $X = Y = A$ and $Z = \lambda$. This gives $Z_X^{\mathcal{C}} = A$ and we can chose $W = A$.

3. Let N_R be a tuple attribute. Then we can assume $X = A(A_1, \ldots, A_n)$, $Y = A(A_1', \ldots, A_n')$ and $Z = A(A_1'', \ldots, A_n'')$ with $A_j \leq A_j', A_j''$ for all $1 \leq j \leq n$. As we have $Z \not\leq Y$ there is at least one i with $A_i'' \not\leq A_i'$. By induction there is a subattribute B_i of A_i' with $B_i \neq \lambda$, such that $(A_i'')_{A_i}^{\mathcal{C}} \leq B_i$ holds.
   Define $B_j = \lambda$ for all other indices j, and take $W = A(B_1, \ldots, B_n) \neq \lambda$. Obviously, then $A_i' \leq B_i$ holds for all $1 \leq i \leq n$, which implies $Y \leq W$. Finally, we have $Z_X^{\mathcal{C}} = A((A_1'')_{A_1'}^{\mathcal{C}}, \ldots, (A_n'')_{A_n'}^{\mathcal{C}}) \leq W$ as derived.

4. Let N_R be a set attribute. In this case we may assume $X = A\{D\}$, $Y = A\{C\}$ and $Z = A\{B\}$ with $D \leq C, B$. As we assume $Z \not\leq Y$ we must have $B \not\leq C$. By induction there is a subattribute $W' \neq \lambda$ of C with $B_D^{\mathcal{C}} \leq W'$. Define $W := A\{W'\}$. Hence, $Y \leq W \neq \lambda$ and $Z_X^{\mathcal{C}} = A\{B_D^{\mathcal{C}}\} \leq A\{W'\} = W$.

□

We conclude this chapter proving the functional dependency of the Higher-Order Entity-Relationship Model.

Theorem 5.10. *Let R be a relationship type with coresponding nested attribute N_R. Let dep be a set of functional dependencies on SubAttr(X). Let Ax be the set of Armstrong-axioms for functional dependencies im HERM. Then we have $dep^* = dep^+$, where the hull dep^+ is built with respect to Ax. For each proper subset of Ax this is no longer the case.*

Proof. In Proposition 5.6 we already showed the correctness of the Armstrong-axioms, i.e., $dep^+ \subseteq dep^*$. So only the other part of the inclusion, i.e., $dep^* \subseteq dep^+$ remains to be proven.

For this assume that $X \to Y$ is functional dependency with $X \to Y \notin dep^+$.

Let $\overline{X}$ be the closure of X as defined in Definition 5.8. We define a relationship set r such that for all relationships $t_1, t_2 \in r$ we have $t_1 \mid_{\overline{X}} = t_2 \mid_{\overline{X}}$ and $t_1 \mid_Y \neq t_2 \mid_Y$ for all subattributes $Y \in \mathrm{SubAttr}(N_R)$ with $\overline{X}_{N_R}^{\mathcal{C}} \leq Y$.

We are able to define such a relationship due to the fact that each domain of a base type has at least two elements. In fact, we consider the maximal subattributes W of $\overline{X}^{\mathcal{C}}_{N_R}$ with $W \neq \lambda$. We choose one value for t_1, another one for t_2 and construct t_1, t_2 from this basis. We have $\overline{X} \leq X$, which implies $t_1 \mid_X = t_2 \mid_X$. On the other hand $X \to Y \notin dep^+$ implies $\overline{X} \not\leq Y$. By Lemma 5.9 we find a subattribute $W \neq \lambda$ of Y with $\overline{X}^{\mathcal{C}}_{N_R} \leq W$. This gives $t_1 \mid_W \neq t_2 \mid_W$ and thus also $t_1 \mid_Y \neq t_2 \mid_Y$. This means that r does not satisfy $X \to Y$, i.e., $\not\models_r X \to Y$.

We now prove $\models_r dep$. For this let $U \to V \in dep$. We consider two cases:

1. If $\overline{X} \not\leq U$ holds, we apply Lemma 5.9. According to this, there is a $W \neq \lambda$ such that $U \leq W$ and $\overline{X}^{\mathcal{C}}_{N_R} \leq W$. Due to the construction of r we have $t_1 \mid_W \neq t_2 \mid_W$, which implies $t_1 \mid_U \neq t_2 \mid_U$. So $\models_r U \to V$ holds.

2. If $\overline{X} \leq U$ holds, the construction of r implies $t_1 \mid_U = t_2 \mid_U$. Using the union rule we have $X \to \overline{X} \in dep^+$. Due to the reflexivity axiom we have $\overline{X} \to U \in dep^+$. Applying the transitivity rule results in $X \to U \in dep^+$ and with $U \to V \in dep$ we also obtain $X \to V \in dep^+$. Due to the definition of the closure this implies $\overline{X} \leq V$. Thus, we have $t_1 \mid_V = t_2 \mid_V$, i.e., $\models_r U \to V$.

So we have $\models_r dep$. From the definition of $dep^* = \{\psi \mid dep \models \psi\}$, we conclude $\models_r dep^*$. Thus, r is a model for dep^*, but not for $X \to Y$. This implies $X \to Y \notin dep^*$, which completes the proof.

The minimality of the set of axioms and rules follows from the known fact that already for flat attributtes none of these rules can be omitted without destroying completeness.

□

Chapter 6

Summary

In this thesis (Diplomarbeit) we considered functional dependencies in the Higher-Order Entity Relationship Model (HERM). One of the major challenge of HERM is the fact that we do not just have flat attributes, but nested attributes using tuple and set constructors. Based on the fact that functional dependencies are axiomatisable in the Relational Database Model using the Armstrong-axioms, our investigation aimed at extending the Armstrong-axioms to HERM.

We found similar structures for the attributes in the RDM and the HERM. We defined a partial order $\leq$ on nested attributes in HERM. This order is similar to the set inclusion $\subseteq$ on sets of flat attributes in the RDM. In fact, we could show that the set of subattributes associated with an entity- or relationship type in a HERM schema carries the structure of a Heyting algebra with respect to this partial order.

Then we tailored Armstrong-axioms from the RDM in such a way that set inclusion $\subseteq$ is replaced by the order $\leq$. This led us to the definition of Armstrong-axioms for functional dependencies im HERM. With this new set of Armstrong-axioms at hand we first showed that we obtained a sound set of axioms and rules.

In a second step, we could prove that this rule system is also complete. Thus, functional dependencies im HERM are axiomatisable. As the total number of functional dependencies on entity or relationship types is finite, the axiomatisation provides a naive way of deciding, whether $X \to Y$ is implied by a set *dep* of functional dependencies. However, it is still necessary to investigate new efficient decision algorithms.

In the following we discuss further problems concerning dependencies in HERM. As we have shown that subattributes carry the structure of a Heyting algebra, we believe to have laid the foundations for these problems to be solved in the same spirit as in this thesis.

- It is natural to extend the axiomatisation of functional dependencies to other classes of dependencies, especially to multi-valued dependencies. This problem has been solved by S. Link and K.-D. Schewe in [LiSc02].

- Another question concerns the problem, whether there are justified normal forms for non-redundancy and absence of anomalies as in the case of the RDM. This is coupled with the question whether effective decompositions into such normal forms can be achieved. First progress in this direction has been made by S. Hartmann and S. Link [HaLi02].
- Computing the hull of a set of dependencies and thus deciding the implication problem can be done more efficiently. The work in [KiLi02] by M. Kirchberg and S. Link contains theoretical and experimental results on this problem.

Bibliography

[Arm74] W.W. Armstrong. Dependency Structures of Database Relationships. Inform. Processing Letters 74 1974, 580-583.

[Bir48] G. Birkhoff. Lattice Theory. American Mathematical Society, 1948, reprinted 1960.

[Che76] P. P.-S. Chen. The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems, 1(1), 1976, 9-36.

[Cod70] E. F. Codd. A Relational Model for Large Shared Data Banks. Communications of the ACM, 13(6), 377-387.

[EoI00] M. Dummett. Elements of Intuitionism. Oxford Science Publications 2^{nd} edition, 2000.

[HaLi02] S. Hartmann, S. Link. Generalising Boyce-Codd Normalform to Conceptual Databases, submitted for publication.

[HuKi87] R. Hull, R. King. Semantic Database Modeling: Survey, Applications and Research Issues. ACM Computing Surveys, Vol.19(3), 1987, 201-260.

[KiLi02] M. Kirchberg, S. Link. On the Implication Problems of Functional Dependencies in the Higher-Order Entity-Realtionship Model, submitted for publication.

[LiSc02] S. Link, K.-D. Schewe. Axiomatizing Functional and Multi-Valued Functional Dependencies in the Higher-Order Entity-Relationship Model, submitted for publication.

[MaRä92] Heikki Mannila, Kari-Jouko Räihä. The Design of Relational Databases. Addison-Wesley, 1992.

[Par89] J. Paredaens, P. De Bra, M. Gyssens, D. Van Gucht. The Structure of the Relational Database Model, EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1989.

[Sch97] Klaus-Dieter Schewe. Datenbanktheorie. Lecture Manuscript. TU Clausthal, Germany, 1997.

[Sch99] Klaus-Dieter Schewe. Datenmodellierung. Lecture Manuscript. TU Clausthal, Germany, 1999.

[Sch02] Klaus-Dieter Schewe. Advanced Database Concepts. Lecture Manuscript. Massey University, New Zealand, Version January 2002.

[Shi81] D. Shipman. The Functional Data Model and the Data Language DAPLEX. ACM Transactions on Database Systems, 6(1), 1981, 140-173.

[Tha00] Bernhard Thalheim. Entity-Relationship Modeling. Springer, 2000.