

# On the Implication Problem for Functional Dependencies in the Higher-Order Entity-Relationship Model

Markus Kirchberg

Sebastian Link

Department of Information Systems  
Massey University,  
Private Bag 11222, Palmerston North, New Zealand,  
Email: [M.Kirchberg|S.Link]@massey.ac.nz

## Abstract

Functional Dependencies have recently been generalized (Hoffmann, Link, Schewe 2002) to the Higher-Order Entity-Relationship Model (HERM). A sound and complete set of rules for the implication of these functional dependencies has also generalized the well-known Armstrong Axioms from the Relational Data Model (RDM) to the HERM.

This article proposes algorithms which solve the implication problem for generalized functional dependencies in the HERM efficiently. Correctness and complexity results are proven. The main result shows that the implication problem for functional dependencies in the HERM is decidable in linear time. Some applications of the implication problem for generalized functional dependencies are discussed. Algorithms are implemented and tested.

**Keywords:** Advanced Foundations of Databases, Logic in Databases, Entity-Relationship Modeling, Functional Dependencies

## 1 Introduction

Functional dependencies are well-known for the Relational Datamodel (RDM) for almost 30 years now. They are used whenever values on some attributes already determine the values on some other attributes. Functional dependencies play an important role in the design theory of relational databases. Fundamental is the finite axiomatization for the implication of functional dependencies by the Armstrong Axioms (Armstrong 1974).

The Higher-Order Entity-Relationship Model, introduced by B. Thalheim in (Thalheim 1992), provides an interesting conceptual datamodel, as it is strictly founded in theory. Schemata in the HERM can be mapped automatically to relational database schemata. It is natural to ask, whether the theory of functional dependencies can be carried over to the conceptual level, i.e., the HERM.

The start to this line of research has been made in (Hoffmann, Link, Schewe 2002) where a finite axiomatization of functional dependencies in the HERM has been proposed. The present article investigates the implication problem for functional dependencies in the HERM. That is, given a set  $\Sigma$  of functional dependencies and a further functional dependency  $\sigma$ , decide whether  $\sigma$  is implied by  $\Sigma$ . The finite axiomatization from (Hoffmann, Link, Schewe 2002) provides a sound and complete set of inference rules  $\mathfrak{R}$  which guarantees that  $\sigma$  is implied by  $\Sigma$  if and only if  $\sigma$  can

be derived from  $\Sigma$  using only rules from  $\mathfrak{R}$ . Finding such a derivation, however, is a difficult problem in general. This paper proposes an algorithm which solves the implication problem for functional dependencies in HERM in linear time. This is of great interest, since the HERM is not only an intuitive tool for conceptual design, but may serve as a mathematical model upon which practical database management systems are built.

The paper repeats fundamental definitions from the HERM in Section 2. One feature of the HERM are nested attributes, which can be derived from flat attributes by various ways of nesting such as a record or finite set construction. Section 3 summarizes the main results from (Hoffmann, Link, Schewe 2002), in particular the sound and complete rule system for the implication of functional dependencies in the HERM. Given a nested attribute  $X$ , the set  $Sub(X)$  of its subattributes carries the structure of a Heyting Algebra  $\mathcal{H}$ . In the beginning of Section 4 it is shown that the implication problem can be solved by forming the closure of a nested attribute with respect to the given set of functional dependencies. This leads to a first algorithm that makes direct use of algebraic operations from  $\mathcal{H}$ . The correctness of this algorithm is proven. In the sequel, some theoretical results reveal how to compute these algebraic operations by familiar means. The first algorithm is revised revealing its complexity. Finally, the correctness of an optimized algorithm is proven. It is shown that it solves the implication problem for functional dependencies in the HERM in linear time. Section 5 comments on the implementation and shows some test results. Some applications of the implication problem for functional dependencies in the HERM are discussed in Section 6. Finally, Section 7 summarizes the results and discusses research plans for the future. In particular, the structure of nested attributes is sufficient to define the concept of functional dependencies for semi-structured data and therefore, for XML as well. We strongly believe that all the results from (Hoffmann, Link, Schewe 2002) and from this paper carry over.

## 2 Preliminaries

We assume familiarity with fundamental definitions of the RDM and functional dependencies in the RDM. Any of (Abiteboul, Hull & Vianu 1995, Paredaens 1989) provide more than we need.

The setting of this paper is the Higher-Order Entity-Relationship Model, introduced in (Thalheim 1992). For a comprehensive study see also (Thalheim 2000). We briefly repeat the most fundamental definitions that are needed for the sequel of the paper.

One key feature of the HERM is the nesting of attributes. Starting point, however, is a set  $\mathbb{B}$  of *base types* such as *STRING*, *INTEGER*, *BOOL*, *DATE* etc., a set  $\mathcal{D}$  of domains and a domain assignment

$dom : \mathbb{B} \rightarrow \mathcal{D}$ . A *type assignment* takes a given countable set  $\mathcal{U}$  of attribute names and assigns a base type to each of these attribute names, i.e.,  $type : \mathcal{U} \rightarrow \mathbb{B}$ .

New types over  $\mathbb{B}$  can be obtained by the application of *type constructors* such as records, finite sets, lists, multisets etc. Throughout the paper we will use the type system  $t := b \mid (a_1 : t_1, \dots, a_n : t_n) \mid \{t\}$ , i.e., a type  $t \in \mathbb{T}$  over  $\mathbb{B}$  is a base type  $b$ , a record type  $(a_1 : t_1, \dots, a_n : t_n)$  with disjoint labels  $a_i \in L$  for  $i = 1, \dots, n$  and a countable set  $L$ , or a finite set type  $\{t\}$ . The results of this paper, however, can be extended to multisets, lists and unions.

The domain assignment  $dom$  for base types can then be extended to a domain assignment  $Dom$  for all types in  $\mathbb{T}$ , i.e.,  $Dom(b) = dom(b)$  for all  $b \in \mathbb{B}$ ,

$$Dom((a_1 : t_1, \dots, a_n : t_n)) = \prod_{i=1}^n \{a_i\} \times Dom(t_i) \text{ and}$$

$Dom(\{t\}) = \mathcal{P}_0(Dom(t))$  where  $\mathcal{P}_0(D)$  denotes the set of all finite subsets of  $D$ . Herein, we make the assumption that every domain of a base type has at least cardinality two.

Given countable disjoint sets  $\mathcal{U}$  and  $L$  of attribute names and labels, respectively, the set  $\mathcal{NA} = \mathcal{NA}(\mathcal{U})$  of *nested attributes* is the smallest set that contains all *flat attributes* in  $\mathcal{U}$ , the *null attribute*  $\lambda$ , the *record-valued attribute*  $X(A_1, \dots, A_n)$  whenever  $X \in L$  and  $A_1, \dots, A_n \in \mathcal{NA}$  are pairwise different, and the *set-valued attribute*  $X\{A\}$  whenever  $X \in L$  and  $A \in \mathcal{NA}$ .

Attributes in which the null attribute  $\lambda$  occurs inside a set-valued attribute have a neat impact on conceptual modeling and do not occur in any other approaches to nested relations. Therefore, the HERM-approach to nested attributes is already different from other approaches to nested attributes such as (Mok, Ng, Embley 1996), (Özsoyoglu, Yuan 1987) when we restrict the type system to base, record and finite set types.

The type assignment  $type : \mathcal{U} \rightarrow \mathbb{B}$  can be extended to a type assignment  $Type : \mathcal{NA} \rightarrow \mathbb{T}$  by  $Type(\lambda) = OK$ ,  $Type(A) = type(A)$  for every  $A \in \mathcal{U}$ ,  $Type(X(A_1, \dots, A_n)) = (A_1 : Type(A_1), \dots, A_n : Type(A_n))$  and  $Type(X\{A\}) = \{Type(A)\}$ . This induces a domain assignment  $Dom$  on nested attributes with  $Dom(X) = Dom(Type(X))$ . Note that the domain of  $OK$  is some singleton set, for instance  $\{ok\}$ .

In the following, we identify nested attributes up to occurrences of  $\lambda$  within a record and up to the order of the components within a record. Define  $\equiv \subseteq \mathcal{NA} \times \mathcal{NA}$  as the smallest equivalence relation on  $\mathcal{NA}$  with  $A(A_1, \dots, A_{i-1}, \lambda, A_{i+1}, \dots, A_n) \equiv A(A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_n)$ ,  $A(\lambda) \equiv \lambda$ ,  $A(A_1, \dots, A_n) \equiv A(A_{\pi(1)}, \dots, A_{\pi(n)})$  for every permutation  $\pi$  on  $\{1, \dots, n\}$ , and  $A\{B\} \equiv A\{C\}$  if and only if  $B \equiv C$ . As an example, the nested attribute  $A(B(C, \lambda, D), E\{\lambda\}, F(\lambda), \lambda, G(H\{K\}, L))$  is equivalent to  $A(E\{\lambda\}, B(C, D), G(H\{K\}, L))$ . For the sake of simplicity, we write  $\mathcal{NA}$  instead of  $\mathcal{NA}/\equiv$ .

The nesting of attributes induces an ordering on nested attributes. Informally,  $A \leq B$  for  $A, B \in \mathcal{NA}$  if and only if  $B$  comprises less information than  $A$  does. We call  $B$  a *subattribute* of  $A$  if and only if  $A \leq B$  holds. More formally,  $\leq \subseteq \mathcal{NA} \times \mathcal{NA}$  is defined as smallest partial order with

- $A \leq \lambda$  for all  $A \in \mathcal{NA}$ ,
- $A(A_1, \dots, A_n) \leq A(A'_1, \dots, A'_m)$  whenever  $A_i \leq A'_i$  for all  $i = 1, \dots, m$ , and
- $A\{B\} \leq A\{C\}$  whenever  $B \leq C$ .

The informal description of a subattribute is formally documented by the following fact. If  $A \leq A'$

holds, then there exists a projection function  $\pi_{A'}^A : Dom(A) \rightarrow Dom(A')$ .

The order  $\leq$  on nested attributes allows to generalize the concept of a subset. A subset  $Y = \{A_1, \dots, A_n\} \subseteq \mathcal{NA}$  is called a *generalized subset* of a finite set  $X \subseteq \mathcal{NA}$  if and only if there is a  $Z = \{A'_1, \dots, A'_n\} \subseteq X$  with  $A'_i \leq A_i$  for all  $i = 1, \dots, n$ .

We are finally prepared to introduce extended database types in HERM. A *database type*  $R = (comp(R), attr(R), id(R))$  of order  $i$  consists of a name  $R$ , a finite set  $comp(R) = \{r_1 : R_1, \dots, r_n : R_n\}$  with pairwise distinct role names  $r_1, \dots, r_n$  and names of database types  $R_1, \dots, R_n$  of order  $j < i$  and at least one  $R_i$  is the name of a database type of order  $i - 1$ , a finite set  $attr(R) \subseteq \mathcal{NA}$  of nested attributes and a finite set  $id(R)$  of keys. Every key  $k \in id(R)$  is of form  $k = comp'(R) \cup X$  where  $comp'(R) \subseteq comp(R)$  and  $X$  is a generalized subset of  $attr(R)$ . Database types  $E$  of order 0 have  $comp(E) = \emptyset$  and are called *entity types*. Database types of order  $i > 0$  are called *relationship types*.

Given an entity type  $E = (\{A_1, \dots, A_n\}, \{B_1, \dots, B_m\})$  we define the *corresponding nested attribute of  $E$*  as  $N_E = E(A_1, \dots, A_n) \in \mathcal{NA}$  and the primary key of  $N_E$  as  $N_E^P = E(B_1, \dots, B_m)$  with  $N_E \leq N_E^P$ . Given a relationship type  $R = (\{r_1 : R_1, \dots, r_k : R_k\}, \{A_1, \dots, A_u\}, \{s_1 : S_1, \dots, s_l : S_l, B_1, \dots, B_v\})$ , the *corresponding nested attribute of  $R$*  is  $N_R = R(r_1(N_{R_1}), \dots, r_k(N_{R_k}), A_1, \dots, A_u) \in \mathcal{NA}$  and the primary key of  $N_R$  as  $N_R^P = R(s_1(N_{S_1}), \dots, s_l(N_{S_l}), B_1, \dots, B_v)$  with  $N_R \leq N_R^P$ .

Finally, a *Higher-Order Entity-Relationship Model Schema* (HERM Schema) is a finite and non-empty set  $\mathcal{S}$  of database types for which for all relationship types  $R \in \mathcal{S}$  and for all  $(r' : R') \in comp(R)$  also  $R' \in \mathcal{S}$  holds. An *instance  $\mathcal{I}$  of  $\mathcal{S}$*  assigns to every database type  $R \in \mathcal{S}$  a finite set  $\mathcal{I}(R) \subseteq Dom(N_R)$  such that for all  $(r : R') \in comp(R)$  and for all  $t \in \mathcal{I}(R)$  we have  $\pi_{R(r(N_{R'}))}^{N_R}(t) \in \mathcal{I}(R')$ , and for every  $k \in id(R)$ , the projection function  $\pi_{N_R^P}^{N_R} |_{\mathcal{I}(R)}$  is injective.

### 3 Axiomatizing Functional Dependencies in HERM

This section summarizes the results from (Hoffmann, Link, Schewe 2002). Fix a set  $\mathcal{U}$  of attribute names and a set  $\mathbb{B}$  of base types together with a type assignment  $type$ .

**Definition 3.1.** Let  $X \in \mathcal{NA}$  be a nested attribute. The set *Sub( $X$ )* of *subattributes of  $X$*  is  $Sub(X) = \{Y \mid X \leq Y\}$ .  $\square$

#### 3.1 The Heyting Algebra of Subattributes

This subsection describes the algebraic structure of  $Sub(X)$ . In fact we study the structure of  $Sub(X)/\equiv$ . Given  $A(A_1, \dots, A_n)$ , it is enough to consider subattributes of the same length, i.e.,  $A(A'_1, \dots, A'_n)$ . For the sake of simplicity, we will omit the quotient  $\equiv$  and use  $Sub(X)$  to refer to  $Sub(X)/\equiv$ . Furthermore, we can always assume that a given nested attribute  $X$  does not contain any occurrence of a  $\lambda$  unless  $X = \lambda$  or the occurrence is of the form  $Y\{\lambda\}$ . Such a representative does always exist within the same equivalence class.

Recall that a *Heyting Algebra* is a distributive lattice  $(L, \sqsubseteq, \sqcap, \sqcup, \Rightarrow, 0)$  with bottom element 0 and a binary operation  $\Rightarrow$  which satisfies:

$$c \sqsubseteq a \Rightarrow b \text{ iff } a \sqcap c \sqsubseteq b$$

for all  $c \in L$ . In this case,  $a \Rightarrow b$  is called the *relative pseudo-complement* of  $a$  with respect to  $b$ . The *pseudo-complement*  $\neg a$  of  $a \in L$  is then defined by  $\neg a = a \Rightarrow 0$ , i.e., as the relative pseudo-complement of  $a$  with respect to the bottom element  $0$ . The top element is given by  $1 = \neg 0$ .

It is obvious that  $(Sub(X), \leq, X, \lambda)$  is a bounded partially ordered set with bottom element  $X$  and top element  $\lambda$ .

**Definition 3.2.** Let  $X \in \mathcal{NA}$  and  $Y, Z \in Sub(X)$ . The *meet*  $Y \bowtie_X Z$ , *join*  $Y \bullet_X Z$  and *relative pseudo-complement*  $Y \rightarrow_X Z$  of  $Y$  and  $Z$  in  $Sub(X)$  are inductively defined as follows:

- $Y \bowtie_X Z = Z$  iff  $Z \leq Y$ ,  $Y \bullet_X Z = Y$  iff  $Z \leq Y$  and  $\lambda \rightarrow_X Z = Z$ , and  $Y \leq Z$  iff  $Y \rightarrow_X Z = \lambda$ ,
- if  $X = A\{B\}$ ,  $Y = A\{C\}$ ,  $Z = A\{D\}$  with  $B \leq C, D$ , then

$$Y \circ_X Z = A\{C \circ_B D\} \quad \text{for } \circ \in \{\bowtie, \bullet\}$$

and if  $Y \not\leq Z$ , then  $Y \rightarrow_X Z = A\{C \rightarrow_B D\}$ .

- if  $X = A(A_1, \dots, A_n)$ ,  $Y = A(A'_1, \dots, A'_n)$  and  $Z = A(A''_1, \dots, A''_n)$ , then  $Y \circ_X Z = A(A'_1 \circ_{A_1} A''_1, \dots, A'_n \circ_{A_n} A''_n)$  for  $\circ \in \{\bowtie, \bullet, \rightarrow\}$ .  $\square$

**Example 3.1.** Assume we pick

$$\begin{aligned} X &= A(A_1(B_1, B_2, B_3\{C_1\}), A_2\{B_4\{C_2(D_1, D_2)\}\}, \\ &\quad A_3\{B_5(C_3\{\lambda\}, C_4)\}) \text{ with} \\ Y &= A(A_1(B_1, B_3\{\lambda\}), A_3\{B_5(C_4)\}) \text{ and} \\ Z &= A(A_1(B_2, B_3\{C_1\}), A_2\{B_4\{\lambda\}\}, A_3\{B_5(C_4)\}). \end{aligned}$$

Applying Definition 3.2 leads to

$$\begin{aligned} Y \bowtie_X Z &= A(A_1(B_1, B_2, B_3\{C_1\}), A_2\{B_4\{\lambda\}\}, \\ &\quad A_3\{B_5(C_4)\}), \\ Y \bullet_X Z &= A(A_1(B_3\{\lambda\}), A_3\{B_5(C_4)\}), \\ Y \rightarrow_X Z &= A(A_1(B_2, B_3\{C_1\}), A_2\{B_4\{\lambda\}\}). \end{aligned}$$

$\square$

The following result describes the algebraic structure of  $Sub(X)$  and has been proven in (Hoffmann, Link, Schewe 2002).

**Theorem 3.1.**  $(Sub(X), \leq, \bowtie_X, \bullet_X, \rightarrow_X, X)$  forms a Heyting-Algebra for every  $X \in \mathcal{NA}$ .  $\square$

### 3.2 Generalized Armstrong Axioms

We repeat the definition of functional dependencies in the HERM, a generalization of the Armstrong Axioms and mention the main result from (Hoffmann, Link, Schewe 2002) that these define a sound and complete set of inference rules for the implication of functional dependencies in the HERM.

**Definition 3.3.** Let  $N \in \mathcal{NA}$  be a nested attribute. A *functional dependency on  $N$*  is an expression of the form  $X \rightarrow Y$  where  $X, Y \in Sub(N)$ . A finite set  $r \subseteq Dom(N)$  satisfies a functional dependency  $X \rightarrow Y$  on  $N$  if and only if for all values  $t_1, t_2 \in r$  with  $\pi_X^N(t_1) = \pi_X^N(t_2)$  always  $\pi_Y^N(t_1) = \pi_Y^N(t_2)$  follows. A functional dependency on a database type  $R$  is a functional dependency on the corresponding nested attribute  $N_R$ .  $\square$

Note that Definition 3.3 includes the definition of functional dependencies from the Relational Data-model for the case of an entity type  $E$  where  $attr(E)$  contains only simple attributes.

The notions of implication ( $\models$ ) and derivability ( $\vdash_{\mathfrak{R}}$ ) with respect to a rule system  $\mathfrak{R}$  for functional dependencies in HERM can be defined analogously to the notions in the RDM, see for instance (Abiteboul

et al. 1995, pp. 164-168). Let  $\Sigma$  be a set of functional dependencies on a database type  $R$ . We are interested in the set of all functional dependencies implied by  $\Sigma$ , i.e.,  $\Sigma^* = \{\varphi \mid \Sigma \models \varphi\}$ . Our aim is finding a rule system  $\mathfrak{R}$  which allows us to show that  $\Sigma^* = \Sigma^+ = \{\varphi \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$ .

**Definition 3.4.** The *generalized Armstrong Axioms for functional dependencies* on a database type  $R$  are

$$\frac{}{X \rightarrow Y} X \leq Y, \quad \frac{X \rightarrow Y}{X \rightarrow X \bowtie_{N_R} Y}, \quad \frac{X \rightarrow Y, Y \rightarrow Z}{X \rightarrow Z}.$$

These rules are called the *reflexivity axiom*, the *extension rule* and the *transitivity rule*.  $\square$

**Example 3.2.** In order to sketch how these rules work we prove the soundness of the meet rule.

$$\frac{\frac{X \rightarrow Y}{X \rightarrow X \bowtie_X Y} \quad \frac{\frac{X \bowtie_X Y \rightarrow X \quad X \rightarrow Z}{X \bowtie_X Y \rightarrow Z}}{X \rightarrow X \bowtie_X Y \rightarrow Z} \quad \frac{X \rightarrow Y \rightarrow Z \rightarrow Y \bowtie_X Z}{X \rightarrow Y \bowtie_X Z}}{X \rightarrow Y \bowtie_X Z}$$

$\square$

The main result of (Hoffmann, Link, Schewe 2002) gives a finite axiomatization for the implication of functional dependencies in the HERM.

**Theorem 3.2.** The generalized Armstrong Axioms are sound and complete for the implication of functional dependencies in the HERM. The generalized Armstrong Axioms are minimal in the sense that completeness cannot be achieved with any of its proper subsets.  $\square$

## 4 Solving the Implication Problem efficiently

The implication problem for functional dependencies in the HERM is to decide whether  $\Sigma \models \sigma$  holds for an arbitrary set  $\Sigma$  of functional dependencies on a database type  $R$  and a further functional dependency  $\sigma$  on  $R$ . Theorem 3.2 shows the equivalence to  $\Sigma \vdash_{\mathfrak{R}} \sigma$  where  $\mathfrak{R}$  are the generalized Armstrong Axioms. This solves the implication problem, however, finding a derivation is arduous. The goal of this article is to solve the problem efficiently. It will turn out that the results from the RDM (Beeri, Bernstein 1979) can be carried over to the HERM.

### 4.1 The Closure

Similar as in the RDM (Beeri, Bernstein 1979), we introduce the notion of a closure for a nested attribute with respect to a given set of functional dependencies.

**Definition 4.1.** Let  $R$  be a database type and  $N_R \leq X$ . Let  $\Sigma$  be a set of functional dependencies on  $R$ . The closure  $\overline{X} \in Sub(N_R)$  of  $X$  with respect to  $\Sigma$  is  $\overline{X} = \bowtie_{N_R} \{Y \mid X \rightarrow Y \in \Sigma^+\}$ .  $\square$

In order to solve the implication problem  $\Sigma \models X \rightarrow Y$  it is sufficient to determine  $\overline{X}$ .

**Proposition 4.1.** Let  $\Sigma$  be a set of functional dependencies on a database type  $R$  and  $X \rightarrow Y$  a functional dependency on  $R$ . Then:

$$X \rightarrow Y \in \Sigma^+ \quad \text{if and only if} \quad \overline{X} \leq Y.$$

*Proof.* If  $X \rightarrow Y \in \Sigma^+$ , then  $Y \in \{Z \mid X \rightarrow Z \in \Sigma^+\}$ . It follows immediately by the property of the meet  $\bowtie_{N_R}$  that  $\overline{X} \leq Y$  holds.

Assume that  $Y \in Sub(\overline{X})$ . It follows that  $Y = A_1 \bowtie_{N_R} \dots \bowtie_{N_R} A_n$  for  $A_1, \dots, A_n \in \{Z \mid X \rightarrow Z \in \Sigma^+\}$ . The meet rule  $\frac{X \rightarrow Y, X \rightarrow Z}{X \rightarrow Y \bowtie_{N_R} Z}$  (Example 3.2) implies  $X \rightarrow Y \in \Sigma^+$ .  $\square$

## 4.2 A first Approach

We will now present a first algorithm which determines the closure  $\overline{X}$  of a given  $X \in \text{Sub}(N_R)$  with respect to a given set  $\Sigma$  of functional dependencies on  $R$ . Our algorithm is similar to the one proposed in (Beeri, Bernstein 1979) for solving the implication problem in the RDM.

**Algorithm 4.1 (Nested Attribute Closure I).**

**Input:**  $X \in \text{Sub}(N_R)$ , set  $\Sigma$  of functional dependencies on  $R$

**Output:** closure  $\overline{X}$  of  $X$  with respect to  $\Sigma$

**Method:**

```

VAR  $X_{\text{old}}, X_{\text{new}}, X^+ \in \text{Sub}(N_R)$ ;
 $X_{\text{new}} := X$ ;
REPEAT
   $X_{\text{old}} := X_{\text{new}}$ ;
  FOR each  $U \rightarrow V \in \Sigma$  DO
    IF  $X_{\text{new}} \leq U$  THEN
       $X_{\text{new}} := X_{\text{new}} \bowtie_{N_R} V$ ;
    ENDIF;
  ENDDO;
UNTIL  $X_{\text{new}} := X_{\text{old}}$ ;
 $X^+ := X_{\text{new}}$ ;
RETURN( $X^+$ );

```

□

We show that Algorithm 4.1 is correct, i.e.,  $\overline{X} = X^+$ .

**Proposition 4.2.** *Algorithm 4.1 is correct.*

*Proof.* We first show that  $\overline{X} \leq X^+$  holds. The algorithm starts with  $X_{\text{new}} := X$  and  $\overline{X} \leq X$  holds obviously. The REPEAT loop computes the meet of  $X_{\text{new}}$  and  $V$  for which  $U \rightarrow V \in \Sigma$  and  $X_{\text{new}} \leq U$  holds.  $U$  must therefore be a subattribute of  $\overline{X}$  and Proposition 4.1 implies  $X \rightarrow U \in \Sigma^+$ . We conclude  $X \rightarrow V \in \Sigma^+$  by transitivity and  $\overline{X} \leq V$ . We have shown that  $\overline{X} \leq X_{\text{new}}$  is invariant for the REPEAT loop, hence  $\overline{X} \leq X^+$ .

We show now that  $X^+ \leq \overline{X}$  holds as well. Since the definition of  $\overline{X}$  depends on  $\Sigma^+$ , we assume that there is a chain

$$\Sigma = \Sigma_0 \subseteq \Sigma_1 \subseteq \dots \subseteq \Sigma^+$$

where every  $\Sigma_i$  results from  $\Sigma_{i-1}$  by application of some derivation rule of the generalized Armstrong Axioms. Let  $\Sigma_i$  be arbitrary with  $Y \rightarrow Z \in \Sigma_i$  and  $X^+ \leq Y$ . We show that  $X^+ \leq Z$  holds in this case. In particular,  $X^+ \leq Z$  follows from  $Y \rightarrow Z \in \Sigma^+$  with  $X^+ \leq Y$ . We obtain  $X^+ \leq \overline{X}$  by using  $Y = X$  and  $Z = \overline{X}$ . For what remains to prove we proceed by induction on  $i$ . If  $i = 0$ , we assume that  $Y \rightarrow Z \in \Sigma$ . If  $X^+ \leq Y$ , then  $X_{\text{new}} \leq Y$  at some point. The REPEAT loop computes  $X_{\text{new}} := X_{\text{new}} \bowtie_{N_R} Z$  and we obtain  $X^+ \leq Z$  as stated. If  $i > 0$ , then  $\Sigma_i - \Sigma_{i-1}$  contains exactly one  $Y \rightarrow Z$ . There is nothing to show for all functional dependencies in  $\Sigma_{i-1}$  (hypothesis). Thus, we consider only  $Y \rightarrow Z$  and distinguish three cases.

- If  $Y \rightarrow Z$  results from applying the reflexivity axiom, then  $Y \leq Z$ . Since  $X^+ \leq Y$  by assumption, we obtain  $X^+ \leq Z$  immediately.

- If  $Y \rightarrow Z$  results from applying the extension rule, then  $Z = Y \bowtie_{N_R} U$  with  $Y \rightarrow U \in \Sigma_{i-1}$ . But then  $X^+ \leq Y$  implies  $X^+ \leq U$  by hypothesis and therefore also  $X^+ \leq Z$ .
- If  $Y \rightarrow Z$  results from applying the transitivity rule, then there is some  $U \in \text{Sub}(N_R)$  with  $Y \rightarrow U \in \Sigma_{i-1}$  and  $U \rightarrow Z \in \Sigma_{i-1}$ . If  $X^+ \leq Y$ , then we conclude  $X^+ \leq U$  and  $X^+ \leq Z$  by hypothesis.

□

In order to study the complexity of Algorithm 4.1 we need to comment on how  $X \leq Y$  can be tested for  $X, Y \in \text{Sub}(N_R)$ . Furthermore, it is described how to compute the meet  $X \bowtie_{N_R} Y$ .

## 4.3 Subattribute Basis

We are going to characterize the subattribute relationship between two nested attributes. Therefore, the notion of a subattribute basis is introduced.

**Definition 4.2.** Let  $X \in \mathcal{NA}$ . The *subattribute basis*  $\text{SubB}(X)$  of  $X$  is the smallest set  $\text{SubB}(X) \subseteq \text{Sub}(X)$  which contains  $\lambda$  and where every  $Y \in \text{Sub}(X)$  can be represented as  $Y = \bowtie_X Z$  for some  $Z \subseteq \text{SubB}(X)$ .

□

**Example 4.1.** *Take for instance the nested attribute  $X = A(A_1\{B_1(C_1, C_2\{\lambda\})\}, A_2(B_2, B_3\{C_3\}), A_3)$ , then  $\text{SubB}(X) = \{\lambda, A(A_1\{\lambda\}), A(A_1\{B_1(C_1)\}), A(A_1\{B_1(C_2\{\lambda\})\}), A(A_2(B_2)), A(A_2(B_3\{\lambda\})), A(A_2(B_3\{C_3\})), A(A_3)\}$ .*

□

We will now show some properties of the subattribute basis.

**Proposition 4.3.** *Let  $X, Y \in \mathcal{NA}$ . Then  $X \leq Y$  if and only if  $\text{SubB}(Y) \subseteq \text{SubB}(X)$ .*

*Proof.* Suppose  $X \leq Y$ . Let  $Z \in \text{SubB}(Y)$ . Assume that  $Z \notin \text{SubB}(X)$ . Since  $Z \in \text{Sub}(X)$  we must have  $Z = X_1 \bowtie_X \dots \bowtie_X X_n$  with  $X_i \in \text{SubB}(X)$ ,  $X_i \notin \{\lambda, Z\}$  for  $1 \leq i \leq n$  and  $n \geq 2$ . From  $Y \leq Z$  follows  $Y \leq X_i$  for  $1 \leq i \leq n$ . We conclude that  $X_i = \bowtie_Y \overline{Y}_i$  for some  $\overline{Y}_i \subseteq \text{SubB}(Y)$  for  $1 \leq i \leq n$  by definition of a subattribute basis. Therefore,  $Z = \bowtie_Y Y'$  with  $Y' \subseteq \text{SubB}(Y)$  and  $|Y'| \geq 2$ . This contradicts the minimality of  $\text{SubB}(Y)$  and the fact that  $Z \in \text{SubB}(Y)$ . It follows that  $Z \in \text{SubB}(X)$  and, therefore,  $\text{SubB}(Y) \subseteq \text{SubB}(X)$ .

Suppose now that  $\text{SubB}(Y) \subseteq \text{SubB}(X)$  holds. Since  $Y = \bowtie_Y Z$  for some  $Z \subseteq \text{SubB}(Y)$  and  $\text{SubB}(Y) \subseteq \text{SubB}(X)$ , it follows that  $Y = \bowtie_Y Z$  for some  $Z \subseteq \text{SubB}(X)$ . Moreover, every element in  $Z$  is also a subattribute of  $X$  and, thus,  $Y = \bowtie_X Z$ . Consequently,  $X \leq Y$ .

□

The next lemma can be shown using a simple induction on the structure of  $N$ .

**Lemma 4.1.** *Let  $N \in \mathcal{NA}$  and  $X, Y \in \text{Sub}(N)$ . If  $X \bowtie_N Y \leq Z$  holds, then  $Z = X' \bowtie_N Y'$  for some  $X' \in \text{Sub}(X)$  and some  $Y' \in \text{Sub}(Y)$ .*

□

The next result allows us to form the meet  $X \bowtie_N Y$  by computing the subattribute basis  $\text{SubB}(X)$  and  $\text{SubB}(Y)$  and taking their union.

**Proposition 4.4.** *Let  $N \in \mathcal{NA}$  and  $N \leq X, Y$ . Then  $\text{SubB}(X \bowtie_N Y) = \text{SubB}(X) \cup \text{SubB}(Y)$ .*

*Proof.* From  $X \bowtie_N Y \leq X, Y$  follows  $SubB(X), SubB(Y) \subseteq SubB(X \bowtie_N Y)$  by Proposition 4.3 and, therefore,  $SubB(X) \cup SubB(Y) \subseteq SubB(X \bowtie_N Y)$ . We show that for any  $Z \in Sub(X \bowtie_N Y)$  there is some  $\hat{Z} \subseteq SubB(X) \cup SubB(Y)$  with  $Z = \bowtie_N \hat{Z}$ . It follows then by Definition 4.2 that  $SubB(X \bowtie_N Y) \subseteq SubB(X) \cup SubB(Y)$ . Let  $Z \in Sub(X \bowtie_N Y)$ . Lemma 4.1 implies that  $Z = X' \bowtie_N Y'$  for  $X' \in Sub(X)$  and  $Y' \in Sub(Y)$ . Moreover,  $X' = \bowtie_{X'} \hat{X}$  for some  $\hat{X} \subseteq SubB(X)$  and  $Y' = \bowtie_{Y'} \hat{Y}$  for some  $\hat{Y} \subseteq SubB(Y)$ . Consequently,  $Z = X' \bowtie_N Y' = \bowtie_N \hat{Z}$  for  $\hat{Z} = \hat{X} \cup \hat{Y} \subseteq SubB(X) \cup SubB(Y)$ . Since  $Z \in Sub(X \bowtie_N Y)$  the meet can be taken over  $X \bowtie_N Y$ .  $\square$

For all up-coming computations, we assume that every given nested attribute  $X$  does not contain  $\lambda$  unless it is  $X = \lambda$  or  $\lambda$  occurs in the form  $Y\{\lambda\}$  within  $X$ . Please note that a given  $X$  might even be transformed into such a form and that such a transformation can be achieved in linear time. Algorithm 4.2 computes the subattribute basis  $SubB(X)$  for a given nested attribute  $X$  in time  $\mathcal{O}(|X|)$  where  $|X|$  denotes the number of all symbols used in  $X$ .

**Algorithm 4.2 (Subattribute Basis).**

**Input:**  $X \in Sub(N_R)$  for a database type  $R$

**Output:**  $SubB(X) \subseteq Sub(N_R)$

**Method:**

VAR  $a : \text{STRING}, A_{\text{new}} \in Sub(N_R), A_{\text{prefix}}, A_{\text{suffix}} : \text{STRING}, \text{level}_{\text{old}}, \text{level} : \text{INTEGER};$

```

level:=0; levelold := 0;
SubB(X) := {λ};
WHILE (Select Next Label a in X) DO
  IF (levelold > level) THEN
    Delete last (levelold - level) 'a'[-]-entries
    from the end of Aprefix ('' ∈ {'(', '{'});
    Delete first (levelold - level) ']'-entries
    from the head of Asuffix (]' ∈ {'}', '}');
  ENDF;
  levelold := level;
  IF Type(a)=Record THEN
    Aprefix := Aprefix + a + '(';
    Asuffix := ')' + Asuffix;
  ELSE IF Type(a)=Set THEN
    Aprefix := Aprefix + a + '{';
    Asuffix := '}' + Asuffix;
    Anew := Aprefix + 'λ' + Asuffix;
    Append Anew to SubB(X);
  ELSE IF a!=λ THEN
    Anew := Aprefix + a + Asuffix;
    Append Anew to SubB(X);
  ENDF;
ENDDO;
RETURN(SubB(X));

```

$\square$

The function 'Select Next Label a' searches through the input string for the next attribute label. While searching the counter 'level' is incremented whenever an opening brace or parenthesis is found. In the same way 'level' is decremented when finding a closing brace or parenthesis. 'Type(a)' determines the corresponding type constructor that is used with the label  $a$  by looking at the next symbol. The next example illustrates the algorithm. Since every symbol

from the input  $X$  is read exactly once, the complexity of Algorithm 4.2 is  $\mathcal{O}(|X|)$ .

**Example 4.2.** Take  $X = A(B\{C(D, E\{F\})\}, G)$  as input for Algorithm 4.2. The following table shows the computation steps.

$a$	$A_{\text{prefix}}$	$A_{\text{suffix}}$	level	$A_{\text{new}}$
A	A(	)	1	
B	A(B{	})	2	A(B{λ})
C	A(B{C(	})	3	
D	A(B{C(D	})	3	A(B{C(D)})
E	A(B{C(E{	})	4	A(B{C(E{λ})})
F	A(B{C(E{F	})	4	A(B{C(E{F})})
G	A(	)	1	A(G)

The result is therefore  $SubB(X) = \{\lambda, A(B\{\lambda\}), A(B\{C(D)\}), A(B\{C(E\{\lambda\})\}), A(B\{C(E\{F\})\}), A(G)\}$ .  $\square$

Given a set  $Z \subseteq Sub(N)$  of subattributes, we will now present an algorithm to compute the meet  $\bowtie_N Z \in Sub(N)$ . Algorithm 4.3 has complexity  $\mathcal{O}(|Z| \cdot |N|)$ . If  $Z$  is a subattribute basis of some subattribute of  $N$ , then  $|Z| \leq |N|$  and the time complexity of Algorithm 4.3 is quadratic in  $|N|$ .

**Algorithm 4.3 (Computation Meet).**

**Input:**  $Z \subseteq Sub(N_R)$  for some database type  $R$

**Output:**  $\bowtie_{N_R} Z \in Sub(N_R)$

**Method:**

```

VAR A, Xmeet ∈ Sub(NR), a, b : STRING, levelA :
INTEGER;
Xmeet := λ;
FOR each A ∈ Z DO
  IF Xmeet = λ THEN
    Xmeet := A;
  ELSE
    Select First Label a IN A;
    WHILE (Select Next Label b In Xmeet) DO
      IF a = b THEN
        Select Next Label a In A;
      ELSE IF b Is Last Entry On levelA THEN
        IF b = λ THEN
          IF Type(a)= Flat THEN
            Replace b by a;
          ELSE
            Replace b by a[*];
          ENDF;
        ELSE IF a ≠ λ THEN
          Determine End Of Level levelA in b;
          IF Type(a)= Flat THEN
            Append a To End Of levelA in b;
          ELSE
            Append a[*] To End Of levelA in b;
          ENDF;
        ENDF;
      ENDDO;
    ENDF;
  ENDDO;
RETURN(Xmeet);

```

$\square$

The string ' $a[*]$ ' denotes the attribute label  $a$  together with the whole record or set structure on its nesting depth ( $level_A$ ).

**Example 4.3.** We continue Example 4.2 and take  $Z = SubB(X)$  as input for Algorithm 4.3. The intermediate steps are shown in the following table.

$A$	$X_{meet}$
$\lambda$	$\lambda$
$A(B\{\lambda\})$	$A(B\{\lambda\})$
$A(B\{C\{D\}\})$	$A(B\{C\{D\}\})$
$A(B\{C\{E\{\lambda\}\}\})$	$A(B\{C\{D, E\{\lambda\}\}\})$
$A(B\{C\{E\{F\}\}\})$	$A(B\{C\{D, E\{F\}\}\})$
$A(G)$	$A(B\{C\{D, E\{F\}\}\}, G)$

We obtain  $X_{meet} = X$  as expected.  $\square$

We bring now Algorithms 4.1, 4.2 and 4.3 together.

**Algorithm 4.4 (Nested Attribute Closure II).**

**Input:**  $X \in \mathbf{Sub}(N_R)$ , set  $\Sigma$  of functional dependencies on database type  $R$

**Output:** closure  $\overline{X}$  of  $X$  with respect to  $\Sigma$

**Method:**

```

VAR  $X_{new}, X_{old} \subseteq \mathbf{Sub}(N_R), X^+ \in \mathbf{Sub}(N_R)$ ;
 $X_{new} := \mathbf{Sub}(X)$ ;
FOR each  $U \rightarrow V \in \Sigma$  compute  $\mathbf{Sub}(U)$ ;
REPEAT
   $X_{old} := X_{new}$ ;
  FOR each  $U \rightarrow V \in \Sigma$  DO
    IF  $\mathbf{Sub}(U) \subseteq X_{new}$  THEN
      Compute  $\mathbf{Sub}(V)$ ;
       $X_{new} := X_{new} \cup \mathbf{Sub}(V)$ ;
       $\Sigma := \Sigma - \{U \rightarrow V\}$ ;
    ENDIF;
  ENDDO;
UNTIL  $X_{new} = X_{old}$ ;
 $X^+ := \bowtie_{N_R} X_{new}$ ;
RETURN( $X^+$ );

```

$\square$

Algorithm 4.4 is correct due to Propositions 4.2, 4.3 and 4.4. If a functional dependency  $U \rightarrow V \in \Sigma$  has once been used to extend  $\mathbf{Sub}(X_{new})$ , it can be removed from the set  $\Sigma$ .

**Example 4.4.** Let  $N_R = A(B\{C\}, D(E, F, G\{H\}), K\{L(M, N)\})$  and  $X = A(B\{\lambda\}, D(E), K\{L(M)\})$ . As functional dependencies we take  $\Sigma = \{U_1 = A(D(E, F)) \rightarrow A(K\{L(M, N)\}) = V_1, U_2 = A(K\{\lambda\}) \rightarrow A(B\{\lambda\}, D(F)) = V_2, U_3 = A(B\{\lambda\}, K\{L(N)\}) \rightarrow A(B\{C\}) = V_3, U_4 = A(D(G\{\lambda\})) \rightarrow A(D(G\{H\})) = V_4\}$ . Now we apply Algorithm 4.4 to the input  $X, \Sigma$ . The precomputation yield

- $X_{new} = \{\lambda, A(B\{\lambda\}), A(D(E)), A(K\{\lambda\}), A(K\{L(M)\})\}$ ,
- $\mathbf{Sub}(U_1) = \{\lambda, A(D(E)), A(D(F))\}$ ,
- $\mathbf{Sub}(U_3) = \{\lambda, A(B\{\lambda\}), A(K\{\lambda\}), A(K\{L(N)\})\}$ ,
- $\mathbf{Sub}(U_2) = \{\lambda, U_2\}$  and  $\mathbf{Sub}(U_4) = \{\lambda, U_4\}$

In the first loop  $\mathbf{Sub}(U_2) \subseteq X_{new}$ , therefore, we compute  $\mathbf{Sub}(V_2) = \{\lambda, A(B\{\lambda\}), A(D(F))\}$  and insert  $A(D(F))$  into  $X_{new}$ . The second loop gives first  $\mathbf{Sub}(U_1) \subseteq X_{new}$  and  $\mathbf{Sub}(V_1) = \{\lambda, A(K\{\lambda\}), A(K\{L(M)\}), A(K\{L(N)\})\}$ . Only  $A(K\{L(N)\})$  is added to  $X_{new}$ . We also find  $\mathbf{Sub}(U_3) \subseteq X_{new}$  with  $\mathbf{Sub}(V_3) = \{\lambda, A(B\{\lambda\}), A(B\{C\})\}$ . Consequently  $A(B\{C\})$  is inserted into  $X_{new}$ . Nothing new happens during the last loop and finally,  $X^+ = A(B\{C\}, D(E, F), K\{L(M, N)\})$ .  $\square$

**Proposition 4.5.** The complexity of Algorithm 4.4 is  $\mathcal{O}(|N_R| \cdot |\Sigma| \cdot \min\{|N_R|, |\Sigma|\})$ .

*Proof.* As seen before, it takes  $\mathcal{O}(|N_R|)$  operations to compute  $\mathbf{Sub}(N_R)$  and clearly  $|\mathbf{Sub}(N_R)| \leq |N_R|$ . If  $N_R \leq X$ , then  $|\mathbf{Sub}(X)| \leq |\mathbf{Sub}(N_R)|$

by Proposition 4.3. The precomputation of the subattribute basis  $\mathbf{Sub}(U)$  for every  $U \rightarrow V \in \Sigma$  takes therefore  $\mathcal{O}(|N_R| \cdot |\Sigma|)$  computations.

In each step of the REPEAT  $\dots$  UNTIL loop, the inner FOR-loop is executed exactly  $|\Sigma|$  times. The inclusion test  $\mathbf{Sub}(U) \subseteq X_{new}$  takes at most  $|\mathbf{Sub}(N_R)|$  operations. The same holds for all operations within the IF  $\dots$  THEN branch. Therefore,  $\mathcal{O}(|N_R| \cdot |\Sigma|)$  operations are necessary for the inner FOR-loop.

The REPEAT  $\dots$  UNTIL loop is executed at most  $|\mathbf{Sub}(N_R)|$  times since the subattribute basis cannot have more elements and at least one element is added in each step. Moreover, the REPEAT  $\dots$  UNTIL loop is executed at most  $|\Sigma| + 1$  times as every element in  $\Sigma$  can contribute to the extension of  $\mathbf{Sub}(X_{new})$  at most once. Finally, the computation of  $\bowtie_{N_R} X_{new}$  takes  $\mathcal{O}(|N_R|^2)$  operations.  $\square$

#### 4.4 Optimization

The idea of the optimized Algorithm 4.5 is the following: for every subattribute  $A \in \mathbf{Sub}(N_R)$  one considers functional dependencies  $U \rightarrow V \in \Sigma$  with  $A \in \mathbf{Sub}(U)$ . An array  $IN$  stores this set of dependencies for every element in some  $\mathbf{Sub}(U)$ . Moreover, one considers for every dependency  $U \rightarrow V$  the number of subattributes in  $\mathbf{Sub}(U)$  which have not been added to the subattribute basis of the closure. This is done using an array  $N$ . If  $N = 0$ , every element in  $\mathbf{Sub}(V)$  is added to the subattribute basis. The set  $X_q$  contains subattributes  $A$  which will definitely belong to the subattribute basis of the closure, and  $N(U \rightarrow V)$  is decremented whenever  $A \in \mathbf{Sub}(U)$ .

**Algorithm 4.5 (Optimized NAC).**

**Input:**  $N_R \in \mathcal{NA}$  for database type  $R$ , set  $\Sigma$  of functional dependencies on  $R, X \in \mathbf{Sub}(N_R)$

**Output:** the closure  $X^+$  of  $X$  with respect to  $\Sigma$

**Method:**

```

VAR  $X_q, X' \subseteq \mathbf{Sub}(N_R)$ ,
N: Array  $\Sigma$  of INTEGER,
IN: Array  $N_R$  of sets of functional dependencies;

 $X' := \emptyset; X_q := \mathbf{Sub}(X)$ ;
FOR each  $A \in \mathbf{Sub}(N_R)$  DO
   $IN(A) := \emptyset$ ;
ENDDO;
FOR each  $U \rightarrow V \in \Sigma$  DO
   $N(U \rightarrow V) := |\mathbf{Sub}(U)|$ ;
  IF  $U = \lambda$  THEN
     $X_q := X_q \cup \mathbf{Sub}(V)$ ;
  ELSE
    FOR each  $A \in \mathbf{Sub}(U)$  DO
       $IN(A) := IN(A) \cup \{U \rightarrow V\}$ ;
    ENDDO;
  ENDIF;
ENDDO;
WHILE  $X_q \neq \emptyset$  DO
  SELECT  $A \in X_q$ ;
   $X_q := X_q - \{A\}$ ;
   $X' := X' \cup \{A\}$ ;
  FOR each  $U \rightarrow V \in IN(A)$  DO
     $N(U \rightarrow V) := N(U \rightarrow V) - 1$ ;
    IF  $N(U \rightarrow V) = 0$  THEN
       $X_q := X_q \cup (\mathbf{Sub}(V) - X')$ ;
    ENDIF;
  ENDDO;
ENDDO;
 $X^+ := \bowtie_{N_R} X'$ ;
RETURN( $X^+$ );

```

$\square$

**Proposition 4.6.** *Algorithm 4.5 is correct and its complexity is  $\mathcal{O}(|N_R| \cdot |\Sigma|)$ .*

*Proof.* For the correctness proof we show  $\overline{X} \leq X^+$  and  $X^+ \leq \overline{X}$  (, or equivalently  $SubB(X^+) \subseteq SubB(\overline{X})$  and  $SubB(\overline{X}) \subseteq SubB(X^+)$ ). The first inclusion follows from the initialization of  $X'$  and  $X_q \subseteq SubB(\overline{X})$ . In fact, only subattributes from  $X_q$  are added to  $SubB(X')$ . On the other hand,  $X_q$  is only extended, if  $N(U \rightarrow V) = 0$  for  $SubB(U) \subseteq X_q$ . This is only possible if every subattribute in  $SubB(U)$  is already included in  $X_q$ , i.e.,  $SubB(U) \subseteq SubB(\overline{X})$ . Using the transitivity rule we infer  $SubB(V) \subseteq SubB(\overline{X})$ .

For the inverse inclusion  $SubB(\overline{X}) \subseteq SubB(X^+)$  we consider, as in the proof of Proposition 4.2, a chain

$$\Sigma = \Sigma_0 \subseteq \Sigma_1 \subseteq \dots \subseteq \Sigma^+ ,$$

where every  $\Sigma_i$  results from  $\Sigma_{i-1}$  by application of one rule from the generalized Armstrong Axioms. If  $\Sigma_i$  is such a set and  $Y \rightarrow Z \in \Sigma_i$  with  $X^+ \leq Y$ , we show  $X^+ \leq Z$ . The claim  $X^+ \leq \overline{X}$  follows then as in the proof of Proposition 4.2.

We proceed by induction on  $i$ . If  $i = 0$ , then we assume that  $Y \rightarrow Z \in \Sigma$ . If  $X^+ \leq Y$ , i.e., if  $SubB(Y) \subseteq SubB(X^+)$ , then every subattribute in  $SubB(Y)$  belongs to  $X_q$  at some point in time. This means, the subattribute will be selected during the WHILE loop and  $N(Y \rightarrow Z)$  will be decremented. As this holds for all subattributes in  $SubB(Y)$ , the number  $N(Y \rightarrow Z)$  becomes eventually 0. In this case,  $SubB(Z) - X'$  is added to  $X_q$  and indirectly to  $X'$ . This implies  $SubB(Z) \subseteq SubB(X^+)$  as claimed. For  $i > 0$ , the set  $\Sigma_i - \Sigma_{i-1}$  contains exactly one  $Y \rightarrow Z$ . The statement can then be proven as in the proof of Proposition 4.2.

In order to prove the complexity, we first look at the initialization loop. As a FOR loop it is executed  $|\Sigma|$  times. Building  $SubB(U)$  and calculating  $|SubB(U)|$  takes at most  $2 \cdot |N_R|$  operations together. The inner FOR loop is executed exactly  $|SubB(U)|$  times. Since adding a functional dependency to  $IN(A)$  can be done in constant time if  $IN(A)$  is represented in an appropriate way as a list, and since  $|SubB(U)| \leq |N_R|$  we obtain a complexity of  $\mathcal{O}(|N| \cdot |\Sigma|)$  for the initialization.

The WHILE loop is executed at most  $|SubB(N_R)|$  times since every subattribute can be selected at most once. The inner FOR loop is executed at most  $|\Sigma|$  times. Clearly, if  $N(U \rightarrow V) = 0$ , then all the subattributes in  $SubB(U)$  have been considered and  $U \rightarrow V$  cannot occur any further in the algorithm. Hence, the IF test evaluates to true at most  $|\Sigma|$  times. Since the union operation on  $X_q$  takes  $\mathcal{O}(|N_R|)$  time, it follows that the statement  $X_q := X_q \cup (SubB(V) - X')$  takes on the whole also  $\mathcal{O}(|N_R| \cdot |\Sigma|)$  time. Hence the algorithm has time complexity  $\mathcal{O}(|N_R| \cdot |\Sigma|)$ .  $\square$

We illustrate Algorithm 4.5 with an example.

**Example 4.5.** *Take  $N_R = A(B\{C\}), D(E, F, G\{H\}), K\{L(M, N)\}$ ,  $X = A(B\{\lambda\}), D(E), K\{L(M)\}$  and  $\Sigma = \{U_1 = A(D(G\{\lambda\})) \rightarrow A(D(G\{H\})) = V_1, U_2 = A(K\{\lambda\}) \rightarrow A(B\{\lambda\}) = V_2, U_3 = A(K\{L(N)\}) \rightarrow A(B\{C\}) = V_3, U_4 = A(D(E, F)) \rightarrow A(K\{L(M, N)\}) = V_4, U_5 = \lambda \rightarrow A(D(F)) = V_5\}$ . The precomputations yield the following:*

- $N(U_1 \rightarrow V_1) = 2, N(U_2 \rightarrow V_2) = 2, N(U_3 \rightarrow V_3) = 3, N(U_4 \rightarrow V_4) = 3, N(U_5 \rightarrow V_5) = 1$

- $IN(A(D(G\{\lambda\}))) = \{U_1 \rightarrow V_1\},$   
 $IN(A(K\{\lambda\})) = \{U_2 \rightarrow V_2, U_3 \rightarrow V_3\},$   
 $IN(A(K\{L(N)\})) = \{U_3 \rightarrow V_3\},$   
 $IN(A(D(E))) = \{U_4 \rightarrow V_4\}, IN(A(D(F))) = \{U_4 \rightarrow V_4\}$
- $X_q = \{\lambda, A(B\{\lambda\}), A(D(E)), A(K\{L(M)\}), A(D(F))\}.$

We show the respective values for every run through the WHILE loop:

1.  $A = \lambda, X_q = \{A(B\{\lambda\}), A(D(E)), A(K\{L(M)\}), A(D(F))\}, X^+ = \{\lambda\},$   
 $N(U_1 \rightarrow V_1) = 1, N(U_2 \rightarrow V_2) = 1,$   
 $N(U_3 \rightarrow V_3) = 2, N(U_4 \rightarrow V_4) = 2$
2.  $A = A(B\{\lambda\}), X_q = \{A(D(E)), A(K\{L(M)\}), A(D(F))\}, X^+ = \{\lambda, A(B\{\lambda\})\}$
3.  $A = A(D(E)), X_q = \{A(K\{L(M)\}), A(D(F))\}, X^+ = \{\lambda, A(B\{\lambda\}), A(D(E))\}, N(U_4 \rightarrow V_4) = 1$
4.  $A = A(K\{L(M)\}), X_q = \{A(D(F))\}, X^+ = \{\lambda, A(B\{\lambda\}), A(D(E)), A(K\{L(M)\})\}$
5.  $A = A(D(F)), X_q = \emptyset, X^+ = \{\lambda, A(B\{\lambda\}), A(D(E)), A(K\{L(M)\}), A(D(F))\}, N(U_4 \rightarrow V_4) = 0, X_q = \{A(K\{\lambda\}), A(K\{L(N)\})\}$
6.  $A = A(K\{\lambda\}), X_q = \{A(K\{L(N)\})\}, X^+ = \{\lambda, A(B\{\lambda\}), A(D(E)), A(K\{L(M)\}), A(D(F)), A(K\{\lambda\})\}, N(U_2 \rightarrow V_2) = 0, X_q = \{A(K\{L(N)\})\}, N(U_3 \rightarrow V_3) = 1$
7.  $A = A(K\{L(N)\}), X_q = \emptyset, X^+ = \{\lambda, A(B\{\lambda\}), A(D(E)), A(K\{L(M)\}), A(D(F)), A(K\{\lambda\}), A(K\{L(N)\})\}, N(U_3 \rightarrow V_3) = 0, X_q = \{A(B\{C\})\}$
8.  $A = A(B\{C\}), X_q = \emptyset, X^+ = \{\lambda, A(B\{\lambda\}), A(D(E)), A(K\{L(M)\}), A(D(F)), A(K\{\lambda\}), A(K\{L(N)\}), A(B\{C\})\}$

The final computation results in  $\bowtie_{N_R} X^+ = A(B\{C\}, D(E, F), K\{L(M, N)\})$ .  $\square$

In the literature,  $\mathcal{O}(|N_R| \cdot |\Sigma|)$  is usually considered as the order of the input. From this point of view, Algorithm 4.5 is a linear time algorithm for the computation of the closure of a nested attribute and also the main part of a linear time algorithm for the implication problem for functional dependencies in the HERM.

**Theorem 4.1.** *The implication problem for functional dependencies in the Higher-Order Entity-Relationship Model is decidable in linear time.*  $\square$

## 5 Implementation and Tests

After investigating the implication problem from a theoretical point of view we will now look at it from a practical perspective. Our aims are 1) to demonstrate that the presented algorithms can be implemented, and 2) to evaluate their efficiency and investigate the practical relevance of the complexity results using randomly generated sets of input arguments.

## 5.1 Remarks on the Implementation

This subsection comments on the implementation of the algorithms outlined in Section 4. All algorithms have been implemented in C based on the pseudocode presented.

Algorithm 4.1 computes the closure  $\bar{X}$  of a given subattribute  $X$ . Here, we chose to represent  $X$  as an unbalanced tree. Each path in the tree corresponds to a subattribute of  $X$ . The tree of  $X$  is constructed in a way that it contains a path for every subattribute of  $X$ . Thus, the test whether  $X_{new} \leq U$  holds corresponds to the test whether the tree representing  $U$  is a subtree of the tree of  $X_{new}$ .

If  $X_{new} \leq U$  holds for some  $U \rightarrow V$ , then the meet of  $X_{new}$  and  $V$  must be computed. Having the tree representation of  $X_{new}$  and  $V$  this computation corresponds to extending the tree of  $X_{new}$  with paths that exist in the tree of  $V$  but not in the tree of  $X_{new}$ .

The implementation of Algorithm 4.4 is based on the notion of a subattribute basis. However, we have already used something similar in Algorithm 4.1. Each path in the tree representation of  $X$  corresponds to an element in the subattribute basis of  $X$  and vice versa. In general, Algorithm 4.4 differs from Algorithm 4.1 in three aspects: the precomputation of  $SubB(U)$  for every  $U \rightarrow V \in \Sigma$ , the string representation of all subattributes of  $X$ , and the deletion of functional dependencies  $U \rightarrow V$  from  $\Sigma$  that have been used to extend  $X$ .

Having such a subattribute basis the subattribute test can simply be replaced by a subset test and the meet computation by a union of sets. Both can be implemented efficiently if the subattribute bases are sorted. Therefore, it is necessary that all input arguments satisfy the following requirement: every two entries of the same record-valued subattribute of  $X$  or a functional dependency in  $\Sigma$ , respectively, must appear in the same order as they appear in the corresponding record-valued subattribute of  $N_R$ . We assume that every given argument satisfies this condition. Furthermore, the union of two subattribute bases is required to have the same order as any subattribute basis generated by Algorithm 4.2. This additional property does not increase the complexity of the algorithm.

Algorithm 4.5 adds further enhancements. Mainly, the number of computations in the inner FOR-loop is reduced. On the other hand, this causes a significant increase in the number of precomputations. This algorithm can be implemented under the same conditions as above.

## 5.2 Tests & Results

We will now present some results of the evaluation of the three introduced algorithms which solve the implication problem for generalized functional dependencies in HERM.

First, we outline the aims underlying our evaluation, followed by some remarks on the test system and an overview of the tests presented in this section. Finally, test results are presented, discussed and evaluated.

The evaluation process mainly aims at 1) evaluating the efficiency of the presented algorithms and investigating the practical relevance of the outlined complexity results using randomly generated sets of input arguments, and 2) determining which subroutines/subalgorithms have a positive or negative, respectively, impact on the overall performance of

every algorithm.

The following properties, settings and restrictions of our test system are important for the interpretation of the results.

The test system consists of three modules and allows to predefine test settings via a configuration file. These include specifications of the maximal number of entries per record-valued subattribute, a range for the average number of entries per record-valued subattribute, a range of probabilities for creating a flat subattribute, probabilities for creating a set- or a record-valued subattribute, the probability for inserting a  $\lambda$ , the number of functional dependencies in  $\Sigma$ , the number of non-applicable functional dependencies among all dependencies in  $\Sigma$ , the number of input arguments to be generated, the test mode, the number of how many times a tests should be repeated and so on.

The first module generates sets of input arguments based on the settings in the configuration file. For our tests, all relevant settings for this module have been set to *random*. However, we aim at testing sets of input arguments that are relevant in practice. Thus, despite having all of the relevant settings set to *random*, there is an internal limit for some of them, e.g., the number of functional dependencies is limited to 1000.

The second module executes the algorithms a predefined number of times and checks whether the computed closure is indeed correct. Herein, the same algorithm is executed repeatedly with exactly the same input arguments as some performances might be affected by other processes using the same system resources.

The third module collects test values and computes test results according to the specified test mode. There are four modes: *normal*, *memory usage*, *split times* and an *internal mode* which is not relevant for this paper. Any combination of the four can be used.

We ran three tests with the same sets of input arguments. First, 1000 of those sets have been computed. Then, the first test was run to determine the average execution time for every algorithm. The second test has been used to obtain the average memory usage for every algorithm. This is an aspect that is often not considered when investigating the efficiency of algorithms from a more theoretical perspective. These two tests meet our first objective. Finally, the third test investigated which subroutines have a positive or negative, respectively, impact on the overall performance of every algorithm. Thus, our second objective.

Test results are shown in Figures 1 and 2. Surprisingly, Algorithm 4.1 has the best results. But why is this?

Considering Figure 1, the memory usage seems to have a big influence on the overall results. However, it cannot be the only reason. When considering Figure 2 and the execution time of the NAC Algorithm subroutines, Algorithm 4.1 is still superior. This is due to the representation of the subattributes in the algorithms. As mentioned before, Algorithm 4.1 is using a tree representation which is equivalent to using a subattribute basis. In fact, it turns out to be a much more efficient representation as the one chosen for Algorithms 4.4 and 4.5.

Furthermore, we have to keep in mind that only practically relevant sets of input arguments are considered. Therefore, execution times are quite low. Therefore, the influence of expensive operations, such as access to main memory, string operations etc, have a big impact and cannot be ignored. Complexity

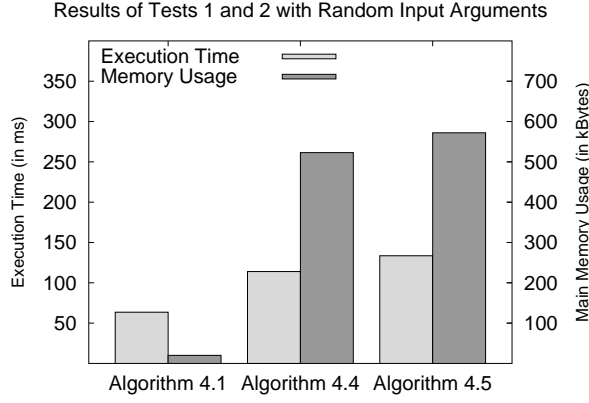


Figure 1: Test Results with Random Arguments

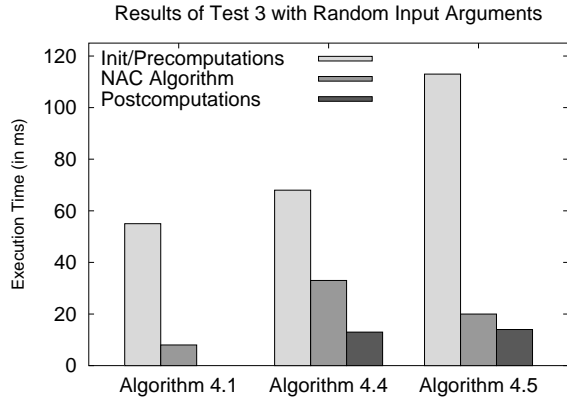


Figure 2: Evaluation Results of the NAC Algorithms

considerations, on the other hand, do not take this into account.

Comparing Algorithms 4.4 and 4.5, Algorithm 4.5 does not really benefit from the enhancements proposed. It has to maintain slightly more memory resulting in a slightly higher average execution time. Considering the individual test results for the 1000 sets of input arguments there is an indication that this changes.

Figure 2 outlines the differences between both algorithms. Algorithm 4.4 has a much faster initialization and precomputation phase than Algorithm 4.5. This is no surprise since Algorithm 4.5 precomputes much more information. The advantage of these increased precomputations come into play when considering the execution time of the algorithm itself. Since Algorithm 4.4 has to compute subattribute bases with every loop, Algorithm 4.5 reuses precomputed information. However, Algorithm 4.5 is not superior due to the fast execution times and the low number of times that one of the precomputed data is reused. Finally, postcomputation times are almost equal.

In summary, the practical evaluation of the algorithms presented has determined that the representation of the data and the size of the memory maintained have a decisive impact on their performance. Complexity considerations have only a minor influence when considering practically relevant sets of input arguments. Therefore, it might be interesting to analyze the complexity from a parameterized point of view. Research in this direction will follow. It would also be interesting to know how both, Algorithm 4.4

and Algorithm 4.5, perform using much more efficient representations. Therefore both, Algorithm 4.2 and Algorithm 4.3, must be modified too.

## 6 Applications of the Implication Problem

The algorithms from Section 4 for solving the implication problem are sufficient to solve several other problems related to functional dependencies. We will demonstrate some applications in this section.

Let  $\Sigma$  be a set of functional dependencies on a database type  $R$  and  $\sigma : X \rightarrow Y \in \Sigma^+$ . A subattribute  $A \in SubB(X)$  with  $A \neq \lambda$  is called *extraneous in  $\sigma$  relative to  $\Sigma$*  if  $X' \rightarrow Y \in \Sigma^+$  with  $X' =_{\bowtie_{N_R}} (SubB(X) - \{A\})$ . If  $\sigma$  does not contain any extraneous subattributes, then it is *reduced*.

In order to test whether  $A \in SubB(X)$  with  $A \neq \lambda$  is extraneous in  $\sigma$ , one can apply Algorithm 4.5 to test if  $X' \rightarrow Y \in \Sigma^+$  for  $X' =_{\bowtie_{N_R}} (SubB(X) - \{A\})$ . A subattribute  $Z$  of  $X$  that functionally determines  $Y$  but does not contain any extraneous subattributes can be found using the following simple procedure:

```

 $X' := SubB(X) - \{\lambda\};$ 
FOR each  $A \in (SubB(X) - \{\lambda\})$  DO
   $Z :=_{\bowtie_{N_R}} (X' - \{A\})$ 
  IF  $Z \rightarrow Y \in \Sigma^+$  THEN
     $X' := X' - \{A\};$ 
  ENDIF;
ENDDO;
 $Z :=_{\bowtie_{N_R}} X';$ 

```

This reduction procedure is effective for functional dependencies in  $\Sigma$  (and those in  $\Sigma^+ - \Sigma$ ). Therefore, it can be used to find a set  $\Sigma_r$  of reduced functional dependencies with  $\Sigma_r^+ = \Sigma^+$ . Since it takes time  $\mathcal{O}(|N_R| \cdot |\Sigma|)$  to detect an extraneous attribute in a single given functional dependency, it takes time  $\mathcal{O}(|N_R|^2 \cdot |\Sigma|)$  to find a reduced functional dependency and time  $\mathcal{O}(|N_R|^2 \cdot |\Sigma|^2)$  to find  $\Sigma_r$ .

Another application of our algorithms is to eliminate redundant functional dependencies. A functional dependency  $\sigma$  is called *redundant* in a set  $\Sigma$  of functional dependencies for a database type  $R$ , if  $(\Sigma - \{\sigma\})^+ = \Sigma^+$ . A *non-redundant cover* of  $\Sigma$  is a set  $\Theta$  of functional dependencies on  $R$  where  $\Theta^+ = \Sigma^+$  and  $\Theta$  does not contain any redundant functional dependency.

In order to determine if  $\sigma$  is redundant in  $\Sigma$ , one can apply Algorithm 4.5 to test if  $\sigma \in (\Sigma - \{\sigma\})^+$ . A subset  $\Theta \subseteq \Sigma$  that is a non-redundant cover of  $\Sigma$  can be found using the following procedure:

```

 $\Theta := \Sigma;$ 
FOR each  $\sigma \in \Sigma$  DO
  IF  $\sigma \in (\Theta - \{\sigma\})^+$  THEN
     $\Theta := \Theta - \{\sigma\};$ 
  ENDIF;
ENDDO;

```

Note that  $\Theta$  is always a subset of  $\Sigma$ , although this is not a requirement of a non-redundant cover. The running time of the covering algorithm is obviously  $\mathcal{O}(|N_R| \cdot |\Sigma|^2)$ .

A nested attribute  $X \in Sub(N_R)$  is called a *superkey* for the database type  $R$ , if there is a functional dependency  $X \rightarrow N_R$  on  $R$ . Hence, a test if  $X \in Sub(N_R)$  is a superkey for  $R$  is equivalent to testing if  $X \rightarrow N_R \in \Sigma^+$ . In order to test whether  $X$  is a *minimal key*, i.e.  $X$  is a superkey and no proper subattribute of  $X$  is a superkey, is equivalent to testing whether  $X \rightarrow N_R$  is reduced.

## 7 Discussion and Future Work

Functional Dependencies are a valuable tool in Relational Database Design. Recently, the notion of functional dependencies and their finite axiomatization have been generalized to a different data model, the Higher-Order Entity-Relationship Model (Hoffmann, Link, Schewe 2002).

The article continues this line of research. It is demonstrated how to solve the implication problem for functional dependencies in the HERM in linear time. This generalizes the solution to the same problem in the RDM.

There are several ways to continue research in this direction. Next we would like to build up a normalization theory in the HERM, similar to the one in the RDM. We are confident that the results for Boyce-Codd Normal Form and Third Normal Form can be generalized to the HERM.

Currently, we are also working on an axiomatization of multi-valued dependencies in the HERM, possibly leading towards a generalization of the Fourth Normal Form.

Looking at nested attributes, we would like to address the finite axiomatization and efficient solution for the implication problem of functional dependencies in XML and semi-structured data.

Finally, the axiomatization of error-robust functional dependencies has recently been studied in the RDM (Link, Schewe, Thalheim 2002) and the HERM (Link, Schewe 2002). These are functional dependencies which are even satisfied after some errors occur. The question is how to solve the implication problem for this class of dependencies efficiently.

## References

- Abiteboul, S., Hull, R. & Vianu, V. (1995), Foundations of Databases, Addison Wesley
- Armstrong, W.W. (1974), Dependency Structures of database relationships, *Inform. Process.* 74, pp. 580-583
- Beeri, C. & Bernstein, P. A. (1979 ), Computational Problems Related to the Design of Normal Form Relational Schemas, *ACM Transactions on Database Systems* , Vol. 4, No. 1, pp. 30-59
- Hoffmann, A., Link, S. & Schewe, K.-D. (2002), Axiomatizing Functional Dependencies in the Higher-Order Entity-Relationship Model, submitted for publication
- Link, S., Schewe, K.-D. & Thalheim, B. (2002), Error-Robust Functional Dependencies, submitted for publication
- Link, S. & Schewe, K.-D. (2002), Axiomatizing Error-Robust Functional Dependencies in the Higher-Order Entity-Relationship Model, submitted for publication
- Mok, W.Y., Ng, Y.K. & Embley, D.W.(1996), A normal form for precisely characterizing redundancy in nested relations, *ACM Transactions on Database Systems*, Vol. 21, 77-106
- Özsoyoglu, Z.M. & Yuan, L.Y.(1987), A new normal form for nested relations, *ACM Transactions on Database Systems*, Vol. 12, 111-136
- Paredaens, J., De Bra, P., Gyssens, M & Van Gucht, D. (1989), The Structure of the Relational Database Model, *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag
- Thalheim, B. (1992), Foundations of entity-relationship modeling, in *Ann.Math.Artificial Intelligence* 6, pp. 1-34
- Thalheim, B. (2000), Entity-Relationship Modeling: Foundations of Database Technology, Springer-Verlag