

# Horn Clauses and Functional Dependencies in Complex-value Databases

Sven Hartmann, Sebastian Link<sup>†</sup>

Information Science Research Centre, Massey University  
Private Bag 11222, Palmerston North, New Zealand  
e-mail: [s.hartmann,s.link]@massey.ac.nz

## Abstract

We extend Fagin's result on the equivalence between functional dependencies in relational databases and propositional Horn clauses. It is shown that this equivalence still holds for functional dependencies in databases that support complex values via nesting of records, lists, sets and multisets.

The equivalence has several implications. Firstly, it extends a well-known result from relational databases to databases which are not in first normal form. Secondly, it characterises the implication of functional dependencies in complex-value databases in purely logical terms. The database designer can take advantage of this equivalence to reduce database design problems to simpler problems in propositional logic. An algorithm is presented for such an application. Furthermore, relational database design tools can be reused to solve problems for complex-value databases.

**Key Words:** Logic in Databases, Functional Dependency, Implication Problem, Complex values, Horn clause

## 1 Introduction

Functional dependencies, first introduced by Codd (Codd 1970), are a fundamental and widely studied concept in relational database theory. The notion of a functional dependency is intuitively simple and is therefore often applied in practice. According to (Delobel & Adiba 1985) about two thirds of all uni-relational dependencies (dependencies over a single relation schema) defined in practice are functional dependencies. It is well-known that in some ways functional dependencies behave precisely the same as a certain well-studied subset of propositional logic. More precisely, Fagin has shown in (Fagin 1977) that a functional dependency  $\sigma$  is a consequence of a set  $\Sigma$  of functional dependencies that all apply to some relation schema if and only if all Horn clauses that correspond to  $\sigma$  are logical consequences of the Horn clauses that correspond to the functional dependencies in  $\Sigma$ .

**Example 1.1.** Consider the relation schema

Copyright ©2006, Australian Computer Society, Inc. This paper appeared at the Seventeenth Australasian Database Conference (ADC), Hobart, Tasmania. Conferences in Research and Practice in Information Technology, Vol. 49. Gill Dobbie and James Bailey, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

<sup>†</sup> Sebastian Link was supported by Marsden Funding, Royal Society of New Zealand

LECTURE = {Class, Lecturer, Time, Room}

together with the following set  $\Sigma$  of functional dependencies on LECTURE:

- Class  $\rightarrow$  Lecturer,
- Class, Time  $\rightarrow$  Room,
- Lecturer, Time  $\rightarrow$  Class, and
- Room, Time  $\rightarrow$  Class.

Suppose we would like to find out whether Room and Time together form a superkey for LECTURE. That is, the functional dependency  $\sigma$ :

Room, Time  $\rightarrow$  Class, Lecturer

is implied by  $\Sigma$ . Fagin's result shows that this decision problem is equivalent to the problem of deciding whether both of the following propositional Horn clauses<sup>1</sup>, represented in implicational form,

- (Room  $\wedge$  Time)  $\Rightarrow$  Class
- (Room  $\wedge$  Time)  $\Rightarrow$  Lecturer

are logically implied by the following set  $\Sigma'$  of propositional Horn clauses:

- Class  $\Rightarrow$  Lecturer,
- (Class  $\wedge$  Time)  $\Rightarrow$  Room,
- (Lecturer  $\wedge$  Time)  $\Rightarrow$  Class,
- (Room  $\wedge$  Time)  $\Rightarrow$  Class.

It is not difficult to see that the answer to these equivalent decision problems is affirmative. To get a little bit more insight into the relationship between Horn clauses and functional dependencies we look at a further example. The functional dependency  $\sigma$

Class, Lecturer, Room  $\rightarrow$  Time

is not implied by  $\Sigma$ . A counterexample to this implication is given, for instance, by the following two tuple relation  $r$

$t_1 = (\text{Databases}, H. \text{ Simpson}, 2:30pm, 3.12),$   
 $t_2 = (\text{Databases}, H. \text{ Simpson}, 4:30pm, 3.12).$

While  $r$  satisfies all functional dependencies in  $\Sigma$  it does not satisfy  $\sigma$ . Let's look at the corresponding problem in terms of Horn clauses. In fact, the truth assignment  $\theta$  that assigns true to the propositional variables Class, Lecturer and Room, and false to the variable Time makes all Horn clauses in  $\Sigma'$  true but leaves the Horn clause

(Class  $\wedge$  Lecturer  $\wedge$  Room)  $\Rightarrow$  Time

<sup>1</sup>The attributes of LECTURE are now used as propositional variables.

false. The point is here that the truth assignment  $\theta$  assigns true to precisely those variables whose corresponding attributes have the same value in the counterexample relation  $r = \{t_1, t_2\}$  above.  $\square$

Due to this result it is possible to take advantage of artificial intelligence research in the area of theorem proving by converting results in that area into results about relational functional dependencies. The equivalence theorem has had extensions to more general classes of dependencies (Sagiv, Delobel, Parker Jr. & Fagin 1981, Fagin 1982) and resulted in several applications (Sagiv 1980, Parker & Delobel 1979, Delobel & Parker 1978).

Many researchers have remarked that classical database design problems need to be revisited in new data formats (Suciu 2001, Vianu 2001, Vincent 1999). Biskup (Biskup 1998) has listed two particular challenges for database design: finding a unifying framework and extending achievements to deal with advanced database features such as complex data constructors. One possibly unifying framework can result from the classification of data models according to the data constructors that are supported by the model. The relational data model can be captured by a single application of the record constructor, arbitrary nesting of record and set constructor covers aggregation and grouping which are fundamental to many semantic data models as well as the nested relational data model (Hull & King 1987, Abiteboul, Hull & Vianu 1995). The Entity-Relationship Model and its extensions require record, set and (disjoint) union constructor (Chen 1976, Thalheim 2000). A minimal set of constructors supported by any object-oriented data model includes records, lists, sets and multisets (Atkinson, Bancilhon, DeWitt, Dittrich, Maier & Zdonik 1989). Genomic sequence data models call for support of records, lists and sets (Li, Ng & Wong 2002). Finally, XML requires at least record (concatenation), list (Kleene Closure), union (optionality), and reference constructor (Bray, Paoli, Sperberg-McQueen, Maler & Yergeau 2004). The following example illustrates the usage of complex data constructors and what type of dependencies may arise between complex data elements and what difficulties they impose.

**Example 1.2.** Consider a simple example of a purchase profile that supermarkets and Online shops may utilise. A single entry consists of the name of the customer, a bag of items the customer bought, and the discount of this purchase received by the customer. Moreover, every item of the customer's bag consists of an article together with the price of that article. A database schema for such an application may look as follows

*Profile(Customer, Bag(Item(Article, Price)), Discount).*

An actual entry in the database may be

*(Homer, ((Chocolate, 3\$), (Chocolate, 3\$), (Beer, 4\$), (Beer, 5\$)), 2\$).*

Suppose that Homer received his discount of 2\$ since beer that costs 4\$ or more is on special. Intuitively, customers with the same bag of items should receive the same discount. This is an actual functional dependency that involves complex data objects, in this case a bag. The presence of the bag constructor shows some surprises. Consider for instance a second data element

*(Bart, ((Chocolate, 4\$), (Chocolate, 5\$), (Beer, 3\$), (Beer, 3\$)), 0\$).*

Bart bought the same bag of articles and has same bag of prices, yet did not receive any discount. This is actually consistent with respect to the functional

dependency since Bart did not have the same bag of items as Homer. In order to receive a discount it matters which articles are bought to which price.  $\square$

The major goal of this paper is to generalise Fagin's Equivalence theorem to databases that support any combination of records, lists, sets and multisets. Our studies will be based on an abstract data model that defines a database schema as an arbitrarily nested attribute where nesting may refer to records, lists, sets and multisets. It is our intention not to focus on the specifics of any particular data model in order to place emphasis on the data constructors themselves. Functional dependencies have previously been defined in terms of subschemata of the underlying database schema (Hartmann, Link & Schewe 2006). Section 6 of (Hartmann et al. 2006) consists of a detailed comparison of our approach to previous work in various concrete data models such as the nested relational data model and XML. In essence, the expressiveness is complementary. Our approach leads to Brouwerian algebras (McKinsey & Tarski 1946) and provides therefore a mathematically well-founded framework that is sufficiently flexible and powerful to study design problems for different classes of constraints with respect to different combinations of data constructors. The presence of the data constructors, in particular that of set and multiset, requires a very detailed analysis in order to generalise the original proof from the relational data model. Most importantly, the results of this paper show that set and bag constructors must be handled differently from record and list constructor in order to capture the semantics of dependencies consistently.

## 2 An Abstract Data Model

Complex-value data models have been proposed to overcome severe limitations of the relational data model when designing many practical database applications (Abiteboul et al. 1995).

### 2.1 Database Schemata

We start with the definition of flat attributes and values for them. A *universe* is a finite set  $\mathcal{U}$  together with domains (i.e. sets of values)  $dom(A)$  for all  $A \in \mathcal{U}$ . The elements of  $\mathcal{U}$  are called *flat attributes*. Flat attributes will be denoted by upper-case characters from the start of the alphabet such as  $A, B, C$  etc.

In the following we will use a set  $\mathcal{L}$  of labels, and assume that the symbol  $\lambda$  is neither a flat attribute nor a label, i.e.,  $\lambda \notin \mathcal{U} \cup \mathcal{L}$ . Moreover, flat attributes are not labels and vice versa, i.e.,  $\mathcal{U} \cap \mathcal{L} = \emptyset$ .

Database schemata in our data model will be given in form of nested attributes. Let  $\mathcal{U}$  be a universe and  $\mathcal{L}$  a set of labels. The set  $\mathcal{NA}(\mathcal{U}, \mathcal{L})$  of *nested attributes over  $\mathcal{U}$  and  $\mathcal{L}$*  is the smallest set satisfying the following conditions:  $\lambda \in \mathcal{NA}(\mathcal{U}, \mathcal{L})$ ,  $\mathcal{U} \subseteq \mathcal{NA}(\mathcal{U}, \mathcal{L})$ , for  $L \in \mathcal{L}$  and  $N_1, \dots, N_k \in \mathcal{NA}(\mathcal{U}, \mathcal{L})$  with  $k \geq 1$  we have  $L(N_1, \dots, N_k) \in \mathcal{NA}(\mathcal{U}, \mathcal{L})$ , for  $L \in \mathcal{L}$  and  $N \in \mathcal{NA}(\mathcal{U}, \mathcal{L})$  we have  $L[N], L\{N\}, L\langle N \rangle \in \mathcal{NA}(\mathcal{U}, \mathcal{L})$ . We call  $\lambda$  *null attribute*,  $L(N_1, \dots, N_k)$  *record-valued attribute*,  $L[N]$  *list-valued attribute*,  $L\{N\}$  *set-valued attribute* and  $L\langle N \rangle$  *multiset-valued attribute*. From now on, we assume that a set  $\mathcal{U}$  of attribute names, and a set  $\mathcal{L}$  of labels is fixed, and write  $\mathcal{NA}$  instead of  $\mathcal{NA}(\mathcal{U}, \mathcal{L})$ . The null attribute  $\lambda$  is a distinguished attribute whose domain is the single null value which indicates that some information exists but has currently been left out. Some detailed explanations for the null attribute  $\lambda$  is offered later on.

**Example 2.1.** The relation schema `LECTURE` with the four flat attributes `Class`, `Lecturer`, `Time` and

Room can be captured by the record-valued nested attribute

$Lecture(Class, Lecturer, Time, Room)$ .

More generally, a relation schema  $R = \{A_1, \dots, A_k\}$  with flat attributes  $A_1, \dots, A_k$  may be viewed as the record-valued attribute  $R(A_1, \dots, A_k)$  using the name  $R$  of the relation schema as a label.  $\square$

**Example 2.2.** Given flat attributes such as Customer, Article, Price and Discount, and labels such as Profile, Item and Bag, we may construct the record-valued attribute

$Item(Article, Price)$ ,

the multiset-valued attribute

$Bag(Item(Article, Price))$ ,

and finally the record-valued attribute

$Profile(Customer, Bag(Item(Article, Price)), Discount)$ .

Using the null attribute  $\lambda$  we may also generate record-valued attributes such as

$Profile(Customer, Bag(Item(Article, \lambda)), \lambda)$

or

$Profile(Customer, Bag(Item(\lambda, \lambda)), Discount)$ .  $\square$

We can extend the mapping  $dom$  from flat attributes to nested attributes, i.e., we define a set  $dom(N)$  of possible data elements for every nested attribute  $N \in \mathcal{NA}$ . For a nested attribute  $N \in \mathcal{NA}$  we define the domain  $dom(N)$  as follows:  $dom(\lambda) = \{ok\}$ ,  $dom(A)$  for  $A \in \mathcal{U}$  as before,  $dom(L(N_1, \dots, N_k)) = \{(v_1, \dots, v_k) \mid v_i \in dom(N_i) \text{ for } i = 1, \dots, k\}$ , i.e., the set of all  $k$ -tuples  $(v_1, \dots, v_k)$  with  $v_i \in dom(N_i)$  for all  $i = 1, \dots, k$ ,  $dom(L\{N\}) = \{\{v_1, \dots, v_n\} \mid v_i \in dom(N) \text{ for } i = 1, \dots, n\}$ , i.e., the set of all finite sets with elements in  $dom(N)$ ,  $dom(L\langle N \rangle) = \{\langle v_1, \dots, v_n \rangle \mid v_i \in dom(N) \text{ for } i = 1, \dots, n\}$ , i.e., the set of all finite multisets with elements in  $dom(N)$ , and  $dom(L[N]) = \{[v_1, \dots, v_n] \mid v_i \in dom(N) \text{ for } i = 1, \dots, n\}$ , i.e., the set of all finite lists with elements in  $dom(N)$ . The empty set, multiset and list are denoted by  $\emptyset$ ,  $\langle \rangle$ , and  $[\ ]$ , respectively. The value  $ok$  can be interpreted as the null value “some information exists, but is currently omitted”.

**Example 2.3.** Consider the nested attribute

$Lecture(Class, Lecturer, Time, Room)$ .

The 4-tuple

$(Databases, H.Simpson, 1pm, 3.12)$

is an element from the domain of this nested attribute. More generally, the domain of a record-valued attribute  $R(A_1, \dots, A_k)$  with flat attributes  $A_1, \dots, A_k$  is the set of all  $k$ -tuples composed out of elements from the corresponding domains  $dom(A_i)$  of the flat attributes  $A_i$ . In other words, the record-valued attribute  $R(A_1, \dots, A_k)$  represents indeed a relation schema.  $\square$

**Example 2.4.** The data element

$(Homer, \langle (Chocolate, 3\$), (Chocolate, 3\$), (Beer, 4\$), (Beer, 5\$), 2\$ \rangle)$

is from the domain of

$Profile(Customer, Bag(Item(Article, Price)), Discount)$ .

Moreover, the data element

$(Homer, \langle (ok, ok), (ok, ok), (ok, ok), (ok, ok), 2\$ \rangle)$ .

is from the domain of

$Profile(Customer, Bag(Item(\lambda, \lambda)), Discount)$ .  $\square$

## 2.2 Subschemas

The replacement of some attribute names by the null attribute  $\lambda$  within a nested attribute decreases the amount of information that is modelled by the corresponding schema. This fact allows to introduce an order between database schemata.

The subattribute relation  $\leq$  on the set of nested attributes  $\mathcal{NA}$  over  $\mathcal{U}$  and  $\mathcal{L}$  is defined by the following rules, and the following rules only:  $N \leq N$ ,  $\lambda \leq A$  for all flat attributes  $A \in \mathcal{U}$ ,  $\lambda \leq N$  for all list-valued attributes  $N$ ,  $L(N_1, \dots, N_k) \leq L(M_1, \dots, M_k)$  whenever  $N_i \leq M_i$  for all  $i = 1, \dots, k$ , and  $L\{N\} \leq L\{M\}$  whenever  $N \leq M$ . For  $N, M$  we say that  $M$  is a subattribute of  $N$  if and only if  $M \leq N$  holds. We write  $M \not\leq N$  if  $M$  is not a subattribute of  $N$ , and  $M < N$  in case  $M \leq N$  and  $M \neq N$ .

**Lemma 2.1.** The subattribute relation is a partial order on nested attributes.  $\blacksquare$

The subattribute relationship between nested attributes generalises the inclusion relationship between sets of attributes in the relational data model.

**Example 2.5.** Consider again the record-valued attribute

$Lecture(Class, Lecturer, Time, Room)$ .

There is a bijection between attribute sets of the relation schema  $Lecture$  and the subattributes of  $Lecture(Class, Lecturer, Time, Room)$ . The empty attribute set  $\emptyset$ , for instance, corresponds to the subattribute

$Lecture(\lambda, \lambda, \lambda, \lambda)$ .

The attribute set  $\{Lecturer, Room\}$  is correspondent to the subattribute

$Lecture(\lambda, Lecturer, \lambda, Room)$ .

We have now seen in several examples that a relation schema is a special case of a database schema represented by a nested attribute. In fact, such a relation schema can be captured by a single application of the record constructor to a finite set of flat attributes.  $\square$

Another example of a subattribute is given by

$Profile(Customer, Bag(Item(\lambda, \lambda)), Discount)$

which is a subattribute of

$Profile(Customer, Bag(Item(Article, Price)), Discount)$ .

Informally,  $M$  is a subattribute of  $N$  if and only if  $M$  comprises at most as much information as  $N$  does. The informal description of the subattribute relation is formally documented by the existence of a projection function  $\pi_M^N : dom(N) \rightarrow dom(M)$  in case  $M \leq N$  holds. For  $M \leq N$  the projection function  $\pi_M^N : dom(N) \rightarrow dom(M)$  is defined as follows:

- if  $N = M$ , then  $\pi_M^N = id_{dom(N)}$  is the identity on  $dom(N)$ ,
- if  $M = \lambda$ , then  $\pi_\lambda^N : dom(N) \rightarrow \{ok\}$  is the constant function that maps  $v \in dom(N)$  to  $ok$ ,
- if  $N = L(N_1, \dots, N_k)$  and  $M = L(M_1, \dots, M_k)$ , then  $\pi_M^N = \pi_{M_1}^{N_1} \times \dots \times \pi_{M_k}^{N_k}$  maps the tuple  $(v_1, \dots, v_k) \in dom(N)$  to  $(\pi_{M_1}^{N_1}(v_1), \dots, \pi_{M_k}^{N_k}(v_k)) \in dom(M)$ ,
- if  $N = L\{N'\}$  and  $M = L\{M'\}$ , then  $\pi_M^N : dom(N) \rightarrow dom(M)$  maps the set  $\{v_1, \dots, v_n\} \in dom(N)$  to  $\{\pi_{M'}^{N'}(v_1), \dots, \pi_{M'}^{N'}(v_n)\} \in dom(M)$ .

- if  $N = L(N')$  and  $M = L(M')$ , then  $\pi_M^N : \text{dom}(N) \rightarrow \text{dom}(M)$  maps the multiset  $\langle v_1, \dots, v_n \rangle \in \text{dom}(N)$  to  $\langle \pi_{M'}^{N'}(v_1), \dots, \pi_{M'}^{N'}(v_n) \rangle \in \text{dom}(M)$ , and
- if  $N = L[N']$  and  $M = L[M']$ , then  $\pi_M^N : \text{dom}(N) \rightarrow \text{dom}(M)$  maps the list  $[v_1, \dots, v_n] \in \text{dom}(N)$  to  $[\pi_{M'}^{N'}(v_1), \dots, \pi_{M'}^{N'}(v_n)] \in \text{dom}(M)$ .

The projection function tells us precisely how to map a database instance of some schema to an instance of any of its subschemata.

**Example 2.6.** Consider the 4-tuple

$$t = (\text{Databases}, H.\text{Simpson}, 1\text{pm}, 3.12)$$

from the domain of

$$N = \text{Lecture}(\text{Class}, \text{Lecturer}, \text{Time}, \text{Room}).$$

The projection  $\pi_X^N(t)$  of  $t$  from  $N$  to the subattribute

$$X = \text{Lecture}(\lambda, \text{Lecturer}, \lambda, \text{Room})$$

is

$$\pi_X^N(t) = (\text{ok}, H.\text{Simpson}, \text{ok}, 3.12). \quad \square$$

**Example 2.7.** The following data element  $t$

$$(\text{Homer}, \langle (\text{Chocolate}, 3\$), (\text{Chocolate}, 3\$), (\text{Beer}, 4\$), (\text{Beer}, 5\$) \rangle, 2\$)$$

from the domain of

$$N = \text{Profile}(\text{Customer}, \text{Bag}(\text{Item}(\text{Article}, \text{Price})), \text{Discount})$$

has projection

$$\pi_X^N(t) = (\text{Homer}, \langle (\text{ok}, \text{ok}), (\text{ok}, \text{ok}), (\text{ok}, \text{ok}), (\text{ok}, \text{ok}) \rangle, 2\$).$$

where

$$X = \text{Profile}(\text{Customer}, \text{Bag}(\text{Item}(\lambda, \lambda)), \text{Discount}). \quad \square$$

### 2.3 Brouwerian algebra of Subattributes

The relational data model is based on the powerset  $\mathcal{P}(R)$  for a relation schema  $R$ . In fact,  $\mathcal{P}(R)$  is a powerset algebra with partial order  $\subseteq$ , set union  $\cup$ , set intersection  $\cap$  and set difference  $-$ . We will now extend these operations to nested attributes. The inclusion order  $\subseteq$  has already been generalised by the subattribute relationship  $\leq$ . The set  $\text{Sub}(N)$  of *subattributes* of  $N$  is  $\text{Sub}(N) = \{M \mid M \leq N\}$ .

We study the algebraic structure of the poset  $(\text{Sub}(N), \leq)$ . A *Brouwerian algebra* (McKinsey & Tarski 1946) is a lattice  $(L, \subseteq, \sqcup, \sqcap, \dashv, 1)$  with top element 1 and a binary operation  $\dashv$  which satisfies  $a \dashv b \subseteq c$  iff  $a \subseteq b \sqcup c$  for all  $c \in L$ . In this case, the operation  $\dashv$  is called the *pseudo-difference*. The *Brouwerian complement*  $\neg a$  of  $a \in L$  is then defined by  $\neg a = 1 \dashv a$ . A Brouwerian algebra is also called a co-Heyting algebra or a dual Heyting algebra. The system of all closed subsets of a topological space is a well-known Brouwerian algebra, see (McKinsey & Tarski 1946). The definition of the subattribute relationship  $\leq$  completely determines the operations of join, meet and pseudo-difference. The following theorem generalises the fact that  $(\mathcal{P}(R), \subseteq, \cup, \cap, -, \emptyset, R)$  is a Boolean algebra for a relation schema  $R$  in the RDM.

**Theorem 2.1.**  $(\text{Sub}(N), \leq, \sqcup_N, \sqcap_N, \dashv_N, N)$  forms a Brouwerian algebra for every  $N \in \mathcal{NA}$ . ■

The nested attribute  $N$  is the top element of  $(\text{Sub}(N), \leq)$ . The *bottom element*  $\lambda_N$  of  $\text{Sub}(N)$  is given by  $\lambda_N = L(\lambda_{N_1}, \dots, \lambda_{N_k})$  whenever  $N = L(N_1, \dots, N_k)$ , and  $\lambda_N = \lambda$  whenever  $N$  is not a record-valued attribute.

In order to simplify notation, occurrences of  $\lambda$  in a record-valued attribute are usually omitted if this does not cause any ambiguities. That is, the subattribute  $L(M_1, \dots, M_k) \leq L(N_1, \dots, N_k)$  is abbreviated by  $L(M_{i_1}, \dots, M_{i_l})$  where  $\{M_{i_1}, \dots, M_{i_l}\} = \{M_j : M_j \neq \lambda_{N_j} \text{ and } 1 \leq j \leq k\}$  and  $i_1 < \dots < i_l$ . If  $M_j = \lambda_{N_j}$  for all  $j = 1, \dots, k$ , then we use  $\lambda$  instead of  $L(M_1, \dots, M_k)$ . The subattribute  $L(A, \lambda)$  of  $L(A, A)$  cannot be abbreviated by  $L(A)$  since this may also refer to  $L(\lambda, A)$ .

**Example 2.8.** The nested attribute

$$\text{Profile}(\text{Customer}, \text{Bag}(\text{Item}(\text{Article}, \lambda)), \lambda)$$

is abbreviated by

$$\text{Profile}(\text{Customer}, \text{Bag}(\text{Item}(\text{Article}))).$$

The nested attribute

$$\text{Profile}(\lambda, \text{Bag}(\text{Item}(\lambda, \lambda)), \text{Discount})$$

is abbreviated by

$$\text{Profile}(\text{Bag}(\lambda), \text{Discount}). \quad \square$$

If the context allows, we omit the index  $N$  from the operations  $\sqcup_N, \sqcap_N, \dashv_N$  and from  $\lambda_N$ .

### 2.4 Order, Multiplicity and The Null Attribute

We give some more explanations on the null attribute  $\lambda$ . From an algebraic point of view it is simply the bottom element  $N \dashv N$  of the Brouwerian algebra carried by  $N$ . As already seen, replacing occurrences of nested attributes by the null attribute according to the rules of the subattribute relationship results in a subattribute and therefore in a decrease of the amount of information that can be modelled. The null attribute therefore allows to obtain different layers of information generating ultimately the structure of a Brouwerian algebra for a fixed database schema. However, the null attribute also offers some interesting features for database modelling, depending on the presence of certain complex objects. Consider for instance the nested attribute  $\text{Speak}(\text{Person}, \text{Foreign}[\text{Language}])$  which is used to store the list of foreign languages a person speaks (ordered according to some preference). Two elements from the corresponding domain could be  $(\text{Bernhard}, [\text{Russian}, \text{English}, \text{French}])$  and  $(\text{John}, [])$ . The projections of these elements on the subattribute  $\text{Speak}(\text{Person}, \text{Foreign}[\lambda])$  are  $(\text{Bernhard}, [\text{ok}, \text{ok}, \text{ok}])$  and  $(\text{John}, [])$  still revealing that Bernhard speaks 3 foreign languages and John speaks none. Suppose that instead of using the list-valued attribute  $\text{Foreign}[\text{Language}]$  we used a set-valued attribute  $\text{Foreign}\{\text{Language}\}$ , i.e., we are only interested in the foreign languages a person speaks, and not in any preferences for these languages. The element  $(\text{Bernhard}, \{\text{Russian}, \text{English}, \text{French}\})$  is mapped to  $(\text{Bernhard}, \{\text{ok}\})$ , and the element  $(\text{John}, \emptyset)$  is mapped to itself. Therefore, the subattribute  $\text{Speak}(\text{Person}, \text{Foreign}\{\lambda\})$  reveals whether a person speaks a foreign language at all. The feature of storing the same data repeatedly therefore enables *counting*, i.e., is supported by lists and multisets, but not by sets.

The second feature is the ability to model order which is supported by lists, but not by sets nor multisets.

This property implies that the projections of any tuple on two subattributes  $X$  and  $Y$  always determine the projection of that tuple on the join  $X \sqcup Y$ . In case of set or multiset constructor this property is not valid anymore. Consider for instance the set-valued attribute  $Duo\{Pair(Girl, Boy)\}$  which represents sets of dancing couples. A tuple might be  $\{(Don\ Quixote, Theresa), (Sancho\ Pansa, Dulcinea)\}$  and the second tuple  $\{(Don\ Quixote, Dulcinea), (Sancho\ Pansa, Theresa)\}$  results from switching partners. Both tuples coincide in their projection on  $Duo\{Pair(\lambda, Boy)\}$  as they evaluate to  $\{(ok, Don\ Quixote), (ok, Sancho\ Pansa)\}$  and coincide in their projection on  $Duo\{Pair(Girl, \lambda)\}$  as they evaluate to  $\{(Dulcinea, ok), (Theresa, ok)\}$ , but they differ on the join  $Duo\{Pair(Girl, Boy)\}$ .

## 2.5 Subattribute Basis

Let  $\mathcal{T}$  be some collection of data constructors and  $N$  an arbitrary nested attribute composed of data constructors in  $\mathcal{T}$  only. What is the minimal set  $\mathfrak{A}_{\mathcal{T}}(N) \subseteq Sub(N)$  such that every element  $t \in dom(N)$  is uniquely determined by its projections  $\{\pi_A^N(t) \mid A \in \mathfrak{A}_{\mathcal{T}}(N)\}$ ?

Consider the simplest of all cases where  $\mathcal{T}$  consists of the record constructor only. Suppose further that nested attributes are generated from flat attributes by a single application of the record constructor. That is,  $N = L(A_1, \dots, A_k)$  for flat attributes  $A_1, \dots, A_k$ . This is just a different notation for the relation schema  $R = \{A_1, \dots, A_k\}$ . Now, every tuple  $t$  over  $R$  (or  $t \in dom(N)$ , respectively) is completely determined by its projections  $\{\pi_{A_1}^N(t), \dots, \pi_{A_k}^N(t)\}$ . In fact, in order to store a tuple we store its values on the individual attributes. From an algebraic point of view the subattributes  $L(A_1, \lambda, \dots, \lambda), \dots, L(\lambda, \dots, \lambda, A_k)$  are the join-irreducible elements of  $(Sub(N), \leq, \sqcup, \sqcap, \lambda_N)$ . Recall that an element  $a$  of a lattice with bottom element  $0$  is called *join-irreducible* if and only if  $a \neq 0$  and if  $a = b \sqcup c$  holds for any elements  $b$  and  $c$ , then  $a = b$  or  $a = c$ . The *subattribute basis* of  $N$ , denoted by  $\mathcal{B}(N)$ , is the set of join-irreducible elements of  $(Sub(N), \leq, \sqcup, \sqcap, \lambda_N)$ . Every element of  $\mathcal{B}(N)$  is called a *basis attribute* of  $N$ . A basis attribute  $X \in \mathcal{B}(N)$  is called *maximal* if and only if  $X \leq Y$  for any basis attribute  $Y \in \mathcal{B}(N)$  implies that  $X = Y$  holds. Basis attributes that are not maximal are called *non-maximal*.

Consider now the case where  $\mathcal{T}$  consists of record and list constructor, i.e.,  $N$  may be generated from flat attributes by finitely many recursive applications of record and list constructor. One can show that for any  $t_1, t_2 \in dom(N)$  and for any  $X, Y \in Sub(N)$  with  $\pi_X^N(t_1) = \pi_X^N(t_2)$  and  $\pi_Y^N(t_1) = \pi_Y^N(t_2)$  also  $\pi_{X \sqcup Y}^N(t_1) = \pi_{X \sqcup Y}^N(t_2)$  holds. That is, the two projections of a tuple on two subattributes  $X$  and  $Y$  uniquely determine the projection of that tuple on the join  $X \sqcup Y$ . This shows, in particular, that  $\mathfrak{A}_{\mathcal{T}}(N)$  is still the set of join-irreducible elements of  $(Sub(N), \leq, \sqcup, \sqcap, \lambda_N)$ .

If we add set or multiset constructor to  $\mathcal{T}$ , then it becomes insufficient to consider join-irreducible elements. In fact, there is some nested attribute  $N$  and distinct elements of  $dom(N)$  which agree on all projections to basis attributes of  $N$ . We have already seen such a nested attribute  $N$  and two such tuples, namely  $Duo\{Pair(Girl, Boy)\}$  and  $\{(Don\ Quixote, Theresa), (Sancho\ Pansa, Dulcinea)\}$  as well as  $\{(Don\ Quixote, Dulcinea), (Sancho\ Pansa, Theresa)\}$ . A further example is the nested attribute  $Bag\{Item(Article, Price)\}$  and the two tuples

(Homer, ((Chocolate, 3\$), (Chocolate, 3\$), (Beer, 4\$), (Beer, 5\$)), 2\$)

and

(Bart, ((Chocolate, 4\$), (Chocolate, 5\$), (Beer, 3\$), (Beer, 3\$)), 0\$).

In the presence of set and multiset constructor we face the difficulty of characterising those pairs of subattributes  $X$  and  $Y$  of  $N$  for which  $t_1, t_2 \in dom(N)$  exist such that  $\pi_X^N(t_1) = \pi_X^N(t_2)$ ,  $\pi_Y^N(t_1) = \pi_Y^N(t_2)$  and  $\pi_{X \sqcup Y}^N(t_1) \neq \pi_{X \sqcup Y}^N(t_2)$ . In other words, what subattributes of  $N$  (other than the basis attributes) are necessary to uniquely determine every single tuple over  $N$ ? What values need to be stored to identify a tuple uniquely? The following definition is used to answer these questions.

**Definition 2.1.** *Let  $N \in \mathcal{NA}$ . The subattributes  $X, Y \in Sub(N)$  are reconcilable if and only if one of the following conditions is satisfied 1)  $Y \leq X$  or  $X \leq Y$ , 2)  $N = L(N_1, \dots, N_k), X = L(X_1, \dots, X_k), Y = L(Y_1, \dots, Y_k)$  where  $X_i$  and  $Y_i$  are reconcilable for all  $i = 1, \dots, k$  or 3)  $N = L[N'], X = L[X'], Y = L[Y']$  where  $X'$  and  $Y'$  are reconcilable.  $\square$*

If  $N$  is  $Duo\{Pair(Girl, Boy)\}$ , and  $X$  and  $Y$  are  $Duo\{Pair(Girl, \lambda)\}$  and  $Duo\{Pair(\lambda, Boy)\}$ , respectively, then  $X$  and  $Y$  are not reconcilable.

**Theorem 2.2.** *Let  $N \in \mathcal{NA}$ . For all  $X, Y \in Sub(N)$  we have that  $X$  and  $Y$  are reconcilable if and only if for all  $t, t' \in dom(N)$  with  $\pi_X^N(t) = \pi_X^N(t')$  and  $\pi_Y^N(t) = \pi_Y^N(t')$  also  $\pi_{X \sqcup Y}^N(t) = \pi_{X \sqcup Y}^N(t')$  holds.  $\blacksquare$*

It still remains to clarify which projections are necessary and sufficient to identify every tuple over a nested attribute generated by finitely many applications of record, list, set and multiset constructor.

**Definition 2.2.** *Let  $N \in \mathcal{NA}$ . The extended subattribute basis  $\mathcal{E}(N) \subseteq Sub(N)$  is the smallest set with the properties that  $\mathcal{B}(N) \subseteq \mathcal{E}(N)$ , and that for all  $X, Y \in \mathcal{E}(N)$  which are not reconcilable also  $X \sqcup Y \in \mathcal{E}(N)$  holds.  $\square$*

The extended subattribute basis is therefore the smallest set that contains the subattribute basis and that is closed under the join of subattributes that are not reconcilable. In the absence of sets and multisets we have  $\mathcal{E}(N) = \mathcal{B}(N)$  since every pair of subattributes is reconcilable. If  $N$  is a set- or multiset-valued attribute, then  $\mathcal{E}(N) = Sub(N)$ . If  $\mathcal{T}$  consists of records, lists, sets and multisets, then  $\mathfrak{A}_{\mathcal{T}}(N) = \mathcal{E}(N)$ . This seems now very natural: for any two subattributes  $X, Y \in \mathcal{E}(N)$  for which the two projections  $\pi_X^N(t)$  and  $\pi_Y^N(t)$  do not determine the value of  $\pi_{X \sqcup Y}^N(t)$ , the subattributes  $X$  and  $Y$  cannot be reconcilable, and  $X \sqcup Y$  is therefore included in  $\mathcal{E}(N)$ .

There is a relatively simple way to reduce the notion of reconcilability to the notion of comparability with respect to  $\leq$ . The idea is to choose the *units*  $U$  of  $N$  such that for all subattributes  $V, W \in Sub(N)$  we have that  $V$  and  $W$  are reconcilable if and only if  $V \sqcap U$  and  $W \sqcap U$  are comparable with respect to  $\leq$  for all units  $U$  of  $N$ . To spell this out, two subattributes  $X, Y$  are comparable with respect to  $\leq$  if and only if  $X \leq Y$  or  $Y \leq X$  holds. This property is achieved by the following definition.

**Definition 2.3.** *Let  $N \in \mathcal{NA}$ . A nested attribute  $U \in \mathcal{NA}$  is a unit of  $N$  if and only if 1)  $U \in Sub(N)$ , and 2)  $\forall X, Y \leq U$  if  $X$  and  $Y$  are reconcilable, then  $X \leq Y$  or  $Y \leq X$ , and 3)  $U$  is  $\leq$ -maximal with the properties 1) and 2). The set of all units of  $N$  is denoted by  $\mathcal{U}(N)$ .  $\square$*

**Example 2.9.** *The units of*

$Profile(Customer, Bag\langle Item(Article, Price)\rangle, Discount)$

are

$$\begin{aligned} U_1 &= Profile(Customer), \\ U_2 &= Profile(Bag\langle Item(Article, Price)\rangle), \text{ and} \\ U_3 &= Profile(Discount). \end{aligned}$$

The two subattributes  $U_1$  and  $U_3$  are reconcilable. In fact, for every  $U \in \{U_1, U_2, U_3\}$  we have  $U_1 \sqcap U \leq U_3 \sqcap U$  or  $U_3 \sqcap U \leq U_1 \sqcap U$ . However, the subattributes

$$\begin{aligned} X &= Profile(Bag\langle Item(Article)\rangle), \text{ and} \\ Y &= Profile(Bag\langle Item(Price)\rangle) \end{aligned}$$

are not reconcilable. In fact,  $X = X \sqcap U_2$  and  $Y = Y \sqcap U_2$  are incomparable with respect to  $\leq$ .  $\square$

## 2.6 Functional Dependencies

The following definition is a natural extension of the definition of FDs from the relational data model.

**Definition 2.4.** Let  $N \in \mathcal{NA}$  be a nested attribute. A functional dependency on  $N$  is an expression of the form  $\mathcal{X} \rightarrow \mathcal{Y}$  where  $\mathcal{X}, \mathcal{Y} \subseteq Sub(N)$  are non-empty. A set  $r \subseteq dom(N)$  satisfies the FD  $\mathcal{X} \rightarrow \mathcal{Y}$  on  $N$ , denoted by  $\models_r \mathcal{X} \rightarrow \mathcal{Y}$ , if and only if for all  $t_1, t_2 \in r$  we have  $\pi_Y^N(t_1) = \pi_Y^N(t_2)$  for all  $Y \in \mathcal{Y}$  whenever  $\pi_X^N(t_1) = \pi_X^N(t_2)$  holds for all  $X \in \mathcal{X}$ .  $\square$

In case a set of subattributes is the singleton  $\{X\}$  we also write simply  $X$  instead of  $\{X\}$ . For  $\mathcal{X} \subseteq Sub(N)$  let  $\vartheta(\mathcal{X}) = \max_{\leq} \{Y \in \mathcal{E}(N) \mid Y \leq X \text{ for some } X \in \mathcal{X}\}$ , i.e.,  $\vartheta(\mathcal{X})$  contains all the extended basis attributes of  $N$  which are subattributes of some element of  $\mathcal{X}$  and which are  $\leq$ -maximal with this property. It is not difficult to see that an instance  $r \subseteq dom(N)$  satisfies  $\mathcal{X} \rightarrow \mathcal{Y}$  if and only if  $r$  satisfies  $\vartheta(\mathcal{X}) \rightarrow \vartheta(\mathcal{Y})$ . Therefore we can assume without loss of generality that every FD  $\mathcal{X} \rightarrow \mathcal{Y}$  is of the form  $\mathcal{X} = \vartheta(\mathcal{X})$  and  $\mathcal{Y} = \vartheta(\mathcal{Y})$ .

**Example 2.10.** Consider again the nested attribute

$Profile(Customer, Bag\langle Item(Article, Price)\rangle, Discount)$ .

Intuitively, the customers who bought the same bag of items should receive the same discount. Formally, this constraint can be specified as the functional dependency

$Profile(Bag\langle Item(Article, Price)\rangle) \rightarrow Profile(Discount)$ .

Note that this FD is completely different from the FD

$\{Profile(Bag\langle Item(Article)\rangle), Profile(Bag\langle Item(Price)\rangle)\} \rightarrow Profile(Discount)$

as demonstrated by the two tuples

$t_1 = (Homer, \langle (Chocolate, 3\$), (Chocolate, 3\$), (Beer, 4\$), (Beer, 5\$)\rangle, 2\$)$

and

$t_2 = (Bart, \langle (Chocolate, 4\$), (Chocolate, 5\$), (Beer, 3\$), (Beer, 3\$)\rangle, 0\$)$ .

In fact,  $\{t_1, t_2\}$  satisfy the first FD, but do not satisfy the second FD.  $\square$

Let  $\Sigma$  be a set of FDs, and  $\sigma$  a single FD, all defined on some nested attribute  $N$ . We say that  $\Sigma$  (finitely) implies  $\sigma$ , denoted by  $\Sigma \models \sigma$  ( $\Sigma \models_{fin} \sigma$ ) if and only if all (finite)  $r \subseteq dom(N)$  that satisfy all FDs in  $\Sigma$  also satisfy  $\sigma$ . Furthermore,  $\Sigma$  implies  $\sigma$  in the world of two-element instances if and only if all  $r = \{t_1, t_2\} \subseteq dom(N)$  that satisfy all dependencies in  $\Sigma$  also satisfy  $\sigma$ . It is not difficult to see that finite and unrestricted implication coincide for FDs. As it

will turn out in this paper, (finite) implication even coincides with implication in the world of two-element instances.

We will now describe the equivalence between the implication of FDs and the logical implication of propositional Horn clauses. To do so we repeat some basic notions regarding boolean propositional logic, and fix some notation.

## 3 The Equivalence

A *literal* is either a propositional variable  $V$  (a positive literal) or the negation  $\neg V$  of a propositional variable  $V$  (a negative literal). A *Horn clause* is a non-empty disjunction of literals, with at most one positive literal. The Horn clause  $\neg V_1 \vee \dots \vee \neg V_m \vee W$  is represented in implicational form as  $V_1 \wedge \dots \wedge V_m \Rightarrow W$  where the empty conjunction is read as *true*. An implicational statement  $V_1 \wedge \dots \wedge V_m \Rightarrow W_1 \wedge \dots \wedge W_n$  with  $n \geq 1$  and positive literals  $W_1, \dots, W_n$  is equivalent to the  $n$  Horn clauses

$$V_1 \wedge \dots \wedge V_m \Rightarrow W_1, \dots, V_1 \wedge \dots \wedge V_m \Rightarrow W_n.$$

Fagin shows in (Fagin 1977) that the FD  $A_1 \dots A_m \rightarrow B_1 \dots B_m$  is a consequence of a set  $\Sigma$  of FDs on a relation schema  $R$  if and only if the corresponding Horn clauses

$$A_1 \wedge \dots \wedge A_m \Rightarrow B_1, \dots, A_1 \wedge \dots \wedge A_m \Rightarrow B_m$$

are logically implied by the corresponding Horn clauses of  $\Sigma$ . The attributes of relation schema  $R$  are therefore interpreted as propositional variables. It is shown that the implication of FDs over arbitrary relations is equivalent to the implication of FDs over two-tuple relations. Given a two-tuple relation over  $R$ , the truth value of a propositional variable  $A$  is assigned *true* if and only if those two tuples agree on the attribute  $A$ .

We would like to generalise this result to FDs in complex-value databases including records, lists, sets and multisets. The presence of these data constructors causes significant problems in generalising the original proof. The first problem is to choose which subattributes of the underlying nested attribute  $N$  are to be interpreted as propositional truth variables. In general, the right choice is to interpret the elements of  $\mathfrak{A}_{\mathcal{T}}(N)$  as propositional variables. That is, the elements of the extended subattribute basis  $\mathcal{E}(N)$  do both, generalise the result by Fagin, and do justice to the presence of sets and multisets.

A second problem is caused by the subattribute relationship  $\leq$ . While attributes in a relation schema form an anti-chain with respect to inclusion, the elements of the extended subattribute basis are partially ordered by  $\leq$ . If two tuples agree on some extended basis attribute  $U$  and  $V \leq U$ , then they will also agree on  $V$ . As the structure of the database schema  $N$  is fixed, this results in a fixed set of Horn clauses that need to be satisfied independently from the given set of constraints. That is, Horn clauses are not only used to encode the dependencies, but also the structure of the underlying database schema.

Given an arbitrary truth assignment to the propositional variables, do there always exist two tuples which precisely agree on those extended basis attributes that are assigned the truth value *true*? While the answer is negative in general, it is actually sufficient to look for an affirmative answer in the presence of those Horn clauses which encode the database schema. It comes then down to showing that there are in general two tuples which precisely agree on

the elements of a  $\leq$ -downward closed set of subattributes which is closed under the join of reconcilable attributes.

Before we describe the equivalence in general we will illustrate the basic ideas on an example. It shows the two different ways of reasoning which we will show to be equivalent.

### 3.1 An Example

Consider the nested attribute

$$N =$$

Dance(Time,Partaker{Name},Duo{Pair(Girl,Boy)},Rating)

together with the following set  $\Sigma$  of FDs

- Dance(Time)  $\rightarrow$   
Dance(Partaker{Name},Duo{Pair(Girl,Boy)},Rating),
- Dance(Partaker{Name})  $\rightarrow$   
{Dance(Duo{Pair(Girl)}),Dance(Duo{Pair(Boy)})},
- {Dance(Duo{Pair(Girl)}),Dance(Duo{Pair(Boy)})}  $\rightarrow$   
Dance(Partaker{Name}), and
- Dance(Duo{Pair(Girl,Boy)})  $\rightarrow$  Dance(Rating).

$N$  models dancing classes in which partakers are grouped into pairs of girls and boys. The first FD says informally that the time of the course determines everything else, i.e., Dance(Time) is a key. The second FD says informally that the set of participants determines the set of boys and the set of girls. Vice versa, the third FD says informally that the set of girls and the set of boys together determine the set of participants. Finally, the last FD says informally that the set of dancing combinations determines the rating. That is, the rating for each class depends on the combination of the dancing partners. Suppose we want to decide if the single FD  $\sigma = \text{Dance(Partaker\{Name\})} \rightarrow \text{Dance(Rating)}$  is a consequence of  $\Sigma$ . The nested two-tuple relation  $r$  consisting of

$$t_1 = (29.2.1600,\{\text{Dulcinea, Theresa, Don Quixote, Sancho}, \\ \{\text{Dulcinea, Don Quixote},(\text{Theresa, Sancho}),10\}) \text{ and} \\ t_2 = (1.3.1600,\{\text{Dulcinea, Theresa, Don Quixote, Sancho}, \\ \{\text{Dulcinea, Sancho},(\text{Theresa, Don Quixote}),3\})$$

satisfies all FDs in  $\Sigma$ , but violates  $\sigma$ . We have therefore found a counterexample relation  $r$ , and  $\Sigma$  does therefore not imply  $\sigma$ . We consider the problem now from a logical point of view. Therefore, the extended basis attributes in  $\mathcal{E}(N)$  are mapped to propositional variables as follows:

- Dance(Time) is  $V_1$ ,
- Dance(Partaker{Name}) is  $V_2$ ,
- Dance(Partaker{\lambda}) is  $V_3$ ,
- Dance(Duo{Pair(Girl,Boy)}) is  $V_4$ ,
- Dance(Duo{Pair(Girl)}) is  $V_5$ ,
- Dance(Duo{Pair(Boy)}) is  $V_6$ ,
- Dance(Duo{\lambda}) is  $V_7$ , and
- Dance(Rating) is  $V_8$ .

The set  $\Pi_N$  of Horn clauses that encodes the structure of the database schema  $N$  is then given by

$$V_2 \Rightarrow V_3, V_4 \Rightarrow V_5, V_4 \Rightarrow V_6, V_5 \Rightarrow V_7, V_6 \Rightarrow V_7.$$

The set  $\Sigma$  of FDs has the following corresponding set  $\Pi$  of Horn clauses

- $V_1 \Rightarrow V_2, V_1 \Rightarrow V_4, V_1 \Rightarrow V_8,$

- $V_2 \Rightarrow V_5, V_2 \Rightarrow V_6,$
- $V_5 \wedge V_6 \Rightarrow V_2,$  and
- $V_4 \Rightarrow V_8$

and  $\sigma$  corresponds to the single Horn clause  $\pi = V_2 \Rightarrow V_8$ . The truth assignment  $\theta$  with  $\theta(V_i) = \text{true}$  iff  $i \in \{2, 3, 5, 6, 7\}$  satisfies all clauses in  $\Pi \cup \Pi_N$ , but violates  $\pi$ . Therefore,  $\pi$  is not a logical consequence of  $\Pi \cup \Pi_N$ . Most importantly, note the correspondence between the nested counterexample relation  $r$  and the truth assignment  $\theta$ . In fact, the tuples  $t_1$  and  $t_2$  agree exactly on their projections to those extended basis attributes whose corresponding propositional variable was assigned the truth value *true* by  $\theta$ . This turns out to be the decisive argument for showing the equivalence.

### 3.2 The Result

Let  $\varphi : \mathcal{E}(N) \rightarrow \mathcal{V}$  denote a bijection between the extended basis attributes of the underlying database schema  $N$  and the set  $\mathcal{V}$  of propositional variables. Consider the FD  $\sigma = \mathcal{X} \rightarrow \mathcal{Y}$  on  $N$  where  $\mathcal{X} = \{X_1, \dots, X_n\}$ . Let  $\Phi(\sigma)$  be the smallest set that contains the Horn clauses  $\varphi(X_1) \wedge \dots \wedge \varphi(X_n) \Rightarrow \varphi(Y)$  for all  $Y \in \mathcal{Y}$ . If  $\Sigma$  is a set of FDs defined on  $N$ , let  $\Pi = \{\Phi(\sigma) \mid \sigma \in \Sigma\}$  denote the corresponding set of Horn clauses over  $\mathcal{V}$ . Furthermore, the set

$$\Pi_N = \{\varphi(U) \Rightarrow \varphi(V) \mid U, V \in \mathcal{E}(N), U \text{ covers}^2 V\}$$

denotes those Horn clauses which encode the structure of  $N$ . For example, the FD

$$\{\text{Dance(Duo\{Pair(Girl)\}),\text{Dance(Duo\{Pair(Boy)\})}\} \rightarrow \text{Dance(Partaker\{Name\})}$$

results in the Horn clause  $V_5 \wedge V_6 \Rightarrow V_2$ . The main result of this paper is the following extension of Fagin's Equivalence Theorem from (Fagin 1977).

**Theorem 3.1.** [Equivalence Theorem] *Let  $N$  be a nested attribute,  $\Sigma$  a set of FDs and  $\sigma$  a single FD on  $N$ . Let  $\Pi_N$  denote the Horn clauses which encode the structure of  $N$ , and  $\Pi$  denote the corresponding set of Horn clauses for  $\Sigma$ . Then*

- 1)  $\Sigma$  implies  $\sigma$ ,
- 2)  $\Sigma$  implies  $\sigma$  in the world of two-tuple instances, and
- 3)  $\Pi \cup \Pi_N$  logically implies  $\pi$  for all  $\pi \in \Phi(\sigma)$

are equivalent. ■

### 3.3 An Outline of the Proof

We will use this subsection to outline the proof of Theorem 3.1. The equivalence between 1) and 2) is not difficult to see.

We will show the equivalence between 2) and 3). First we show that 3) implies 2). Suppose that 2) does not hold, i.e.,  $\Sigma$  does not imply  $\sigma = \mathcal{X} \rightarrow \mathcal{Y}$  in the world of two-tuple instances over  $N$ . That is, there are some  $t_1, t_2 \in \text{dom}(N)$  with  $\models_{\{t_1, t_2\}} \tau$  for all  $\tau \in \Sigma$ , but  $\not\models_{\{t_1, t_2\}} \sigma$ . Lemma 3.1 shows then that  $\models_{\theta_{\{t_1, t_2\}}} \Pi \cup \Pi_N$ , but  $\not\models_{\theta_{\{t_1, t_2\}}} \pi$  for some  $\pi \in \Phi(\sigma)$ , i.e., 3) does not hold neither. In particular,  $\models_{\theta_{\{t_1, t_2\}}} \Pi_N$  for the following reason. Let  $V \Rightarrow W \in \Pi_N$ , i.e.,  $V = \varphi(X)$  and  $W = \varphi(Y)$  with  $X, Y \in \mathcal{E}(N)$  and  $Y \leq X$ . If  $\theta_{\{t_1, t_2\}}(V) = \text{true}$ , then  $\pi_X^N(t_1) = \pi_X^N(t_2)$  and thus

<sup>2</sup> $U$  covers  $V$  iff  $U < V$  and for all  $W \in \mathcal{E}(N)$  with  $U \leq W \leq V$  we have  $U = W$  or  $V = W$ , this is just the standard definition of a *cover relation* for posets, see (Anderson 1987)

$\pi_Y^N(t_1) = \pi_Y^N(t_2)$  since  $Y \leq X$ . This, however, means that  $\theta_{\{t_1, t_2\}}(W) = \text{true}$  holds as well, and therefore  $\models_{\theta_{\{t_1, t_2\}}} V \Rightarrow W$ . To complete the proof for this direction it remains to show the following lemma.

**Lemma 3.1.** *Let  $\sigma$  be an FD on the nested attribute  $N$ , and  $r = \{t_1, t_2\} \subseteq \text{dom}(N)$ . Then  $\models_r \sigma$  if and only if  $\models_{\theta_r} \pi$  for all  $\pi \in \Phi(\sigma)$ , where*

$$\theta_r(V) = \begin{cases} \text{true}, & \text{if } \pi_{\varphi^{-1}(V)}^N(t_1) = \pi_{\varphi^{-1}(V)}^N(t_2) \\ \text{false}, & \text{else} \end{cases}$$

for all  $V \in \varphi(\mathcal{E}(N))$ . ■

In order to complete the proof of Theorem 3.1 it remains to show that 2) implies 3). Suppose that 3) does not hold. We will show that 2) does not hold neither. Since 3) does not hold, there is some truth assignment  $\theta$  which makes every formula in  $\Pi \cup \Pi_N$  true, but makes some  $\pi \in \Phi(\sigma)$  false. It is now sufficient to find some  $r = \{t_1, t_2\} \subseteq \text{dom}(N)$  such that  $\theta = \theta_r$ . In this case, Lemma 3.1 shows that  $\models_r \tau$  for all  $\tau \in \Sigma$  and  $\not\models_r \sigma$ , i.e., 2) does not hold.

Let  $\mathcal{U}(N) = \{U_1, \dots, U_k\}$ , and  $\mathcal{X}_{U_i}^+ = \{X \in \mathcal{E}(N) \mid X \leq U_i \text{ and } \theta(\varphi(X)) = \text{true}\} \cup \{\lambda_N\}$ . Note that  $\mathcal{X}_{U_i}^+$  is closed downwards with respect to  $\leq$  since  $\theta$  satisfies  $\Pi_N$ . Moreover, let  $\mathcal{X}^+ = \{X_1 \sqcup \dots \sqcup X_k \mid X_i \in \mathcal{X}_{U_i}^+ \text{ for } 1 \leq i \leq k\}$ . In this case,  $\mathcal{X}^+$  is non-empty, closed downwards with respect to  $\leq$ , and closed under the join of reconcilable elements.  $\mathcal{X}^+$  is non-empty since  $\lambda_N \in \mathcal{X}_{U_i}^+$  for  $i = 1, \dots, k$ . It is closed downwards with respect to  $\leq$  since every  $\mathcal{X}_{U_i}^+$  has the same property. In order to see that  $\mathcal{X}^+$  is closed under the join of reconcilable elements suppose that  $X = X_1 \sqcup \dots \sqcup X_k$ ,  $X' = X'_1 \sqcup \dots \sqcup X'_k \in \mathcal{X}^+$  are reconcilable. Since  $X \sqcap U_i = X_i$  and  $X' \sqcap U_i = X'_i$  for  $i = 1, \dots, k$  it must be the case that  $X_i$  and  $X'_i$  are comparable with respect to  $\leq$  for all  $i = 1, \dots, k$ . This, however, means that  $X \sqcup X' \in \mathcal{X}^+$  by definition of  $\mathcal{X}^+$ . The following lemma then shows the existence of two tuples  $t_1, t_2 \in \text{dom}(N)$  with the property that for all  $X \in \text{Sub}(N)$  we have  $\pi_X^N(t_1) = \pi_X^N(t_2)$  if and only if  $X \in \mathcal{X}^+$  holds. Since for all  $X \in \mathcal{E}(N)$  we have that  $\theta(\varphi(X)) = \text{true}$  if and only if  $X \in \mathcal{X}^+$  if and only if  $\pi_X^N(t_1) = \pi_X^N(t_2)$  holds, it follows that  $\theta = \theta_{\{t_1, t_2\}}$ . This concludes the proof of Theorem 3.1.

**Lemma 3.2.** *Let  $N \in \mathcal{NA}$ , and  $\emptyset \neq \mathcal{X} \subseteq \text{Sub}(N)$  an ideal with respect to  $\leq$  with the property that for reconcilable  $X, Y \in \mathcal{X}$  also  $X \sqcup Y \in \mathcal{X}$  holds. Then there are  $t_N, t'_N \in \text{dom}(N)$  such that for all  $W \in \text{Sub}(N)$  we have  $\pi_W^N(t_N) = \pi_W^N(t'_N)$  if and only if  $W \in \mathcal{X}$ . ■*

The detailed (and challenging) proof of Lemma 3.2 was previously published (Hartmann et al. 2006).

#### 4 First-Literal Unit Resolution

The Equivalence Theorem 3.1 states that the FD  $\sigma$  is a consequence of the set  $\Sigma$  of FDs on the underlying database schema  $N$  if and only if each of the corresponding Horn clauses in  $\Phi(\sigma)$  is a logical consequence of  $\Pi \cup \Pi_N$ . The last problem, however, can be easily converted into the well-studied problem of satisfiability of propositional Horn clauses (Horn 1951, Henschen & Vos 1974, Dowling & Gallier 1984). A fast algorithm for the Horn clause satisfiability problem is the “first-literal unit resolution procedure” due to Chang (Chang 1976, Chang & Lee 1987). Using the Equivalence Theorem, we exploit Chang’s algorithm

to obtain an efficient algorithm that solves the implication problem for FDs in complex-value databases. In the first step of the algorithm, we form a set  $\mathcal{S}$  of strings of symbols. Each string consists of extended basis attributes, negation signs ( $\sim$ ) and commas ( $,$ ). For each FD

$$\{X_1, \dots, X_n\} \rightarrow \{Y_1, \dots, Y_m\}$$

in  $\Sigma$  we include in  $\mathcal{S}$  the  $m$  strings

$$\begin{aligned} &\sim X_1, \dots, \sim X_n, Y_1 \\ &\quad \vdots \\ &\sim X_1, \dots, \sim X_n, Y_m \end{aligned}$$

The string  $\sim X_1, \dots, \sim X_n, Y_i$  corresponds to the Horn clause

$$\neg\varphi(X_1) \vee \dots \vee \neg\varphi(X_n) \vee \varphi(Y_i).$$

Furthermore, consider each pair  $U, V \in \mathcal{E}(N)$  with  $V < U$  such that for all  $W \in \mathcal{E}(N)$  with  $V \leq W \leq U$  we have  $V = W$  or  $U = W$ . For each of these pairs  $U, V$  we include in  $\mathcal{S}$  the string  $\sim U, V$ . Let  $\sigma$  be the FD  $\mathcal{X} \rightarrow \mathcal{Y}$  where  $\mathcal{X} = \{X_1, \dots, X_n\}$ . For convenience, we assume that  $\mathcal{Y}$  is a singleton. Otherwise, the entire algorithm is repeated for every element of  $\mathcal{Y}$ . Anyway, for  $\mathcal{Y} = \{Y\}$  we include in  $\mathcal{S}$  the  $(n+1)$  strings

$$\begin{aligned} &X_1 \\ &\quad \vdots \\ &X_n \\ &\sim Y \end{aligned}$$

which correspond to the negation of the Horn clause

$$\neg\varphi(X_1) \vee \dots \vee \neg\varphi(X_n) \vee \varphi(Y).$$

As a simple example consider the nested attribute

$$N = \text{Dance}(\text{Time}, \text{Partaker}\{\text{Name}\}, \text{Duo}\{\text{Pair}(\text{Girl}, \text{Boy})\}, \text{Rating})$$

and let  $\Sigma$  and  $\sigma$  be as in Subsection 3.1. In this case,  $\mathcal{S}$  contains the 14 strings:

$$\begin{aligned} &\sim \text{Dance}(\text{Time}), \text{Dance}(\text{Partaker}\{\text{Name}\}) \\ &\sim \text{Dance}(\text{Time}), \text{Dance}(\text{Duo}\{\text{Pair}(\text{Girl}, \text{Boy})\}) \\ &\sim \text{Dance}(\text{Time}), \text{Dance}(\text{Rating}) \\ &\sim \text{Dance}(\text{Partaker}\{\text{Name}\}), \text{Dance}(\text{Duo}\{\text{Pair}(\text{Girl})\}) \\ &\sim \text{Dance}(\text{Partaker}\{\text{Name}\}), \text{Dance}(\text{Duo}\{\text{Pair}(\text{Boy})\}) \\ &\sim \text{Dance}(\text{Duo}\{\text{Pair}(\text{Girl})\}), \sim \text{Dance}(\text{Duo}\{\text{Pair}(\text{Boy})\}), \\ &\quad \text{Dance}(\text{Partaker}\{\text{Name}\}) \\ &\sim \text{Dance}(\text{Duo}\{\text{Pair}(\text{Girl}, \text{Boy})\}), \text{Dance}(\text{Rating}) \\ &\sim \text{Dance}(\text{Partaker}\{\text{Name}\}), \text{Dance}(\text{Partaker}\{\lambda\}) \\ &\sim \text{Dance}(\text{Duo}\{\text{Pair}(\text{Girl}, \text{Boy})\}), \text{Dance}(\text{Duo}\{\text{Pair}(\text{Girl})\}) \\ &\sim \text{Dance}(\text{Duo}\{\text{Pair}(\text{Girl}, \text{Boy})\}), \text{Dance}(\text{Duo}\{\text{Pair}(\text{Boy})\}) \\ &\sim \text{Dance}(\text{Duo}\{\text{Pair}(\text{Girl})\}), \text{Dance}(\text{Duo}\{\lambda\}) \\ &\sim \text{Dance}(\text{Duo}\{\text{Pair}(\text{Boy})\}), \text{Dance}(\text{Duo}\{\lambda\}) \\ &\text{Dance}(\text{Partaker}\{\text{Name}\}) \\ &\sim \text{Dance}(\text{Rating}) \end{aligned}$$

We call each attribute in  $\mathcal{E}(N)$  an “atom” and the concatenation of a negation sign with such an atom a “negative atom”. The algorithm proceeds by searching for an atom  $X$  such that (a)  $X$  is a string in  $\mathcal{S}$ , and (b) There is a string in  $\mathcal{S}$  that begins with  $\sim X$ . In our example the atom  $\text{Dance}(\text{Partaker}\{\text{Name}\})$  satisfies both (a) and (b). If there are several atoms  $X$  that satisfy (a) and (b), the algorithm would now arbitrarily select one of them. In the next step of the algorithm, each string that begins with  $\sim X$  is shortened by erasing the leading negation sign, the  $X$ , and the comma that follows  $X$  (if there is such a comma).

$\sim$  Dance(Time), Dance(Partaker{Name})  
 $\sim$  Dance(Time), Dance(Duo{Pair(Girl,Boy)})  
 $\sim$  Dance(Time), Dance(Rating)  
 Dance(Duo{Pair(Girl)})  
 Dance(Duo{Pair(Boy)})  
 $\sim$  Dance(Duo{Pair(Girl)}),  $\sim$  Dance(Duo{Pair(Boy)}),  
     Dance(Partaker{Name})  
 $\sim$  Dance(Duo{Pair(Girl,Boy)}), Dance(Rating)  
 Dance(Partaker{ $\lambda$ })  
 $\sim$  Dance(Duo{Pair(Girl,Boy)}), Dance(Duo{Pair(Girl)})  
 $\sim$  Dance(Duo{Pair(Girl,Boy)}), Dance(Duo{Pair(Boy)})  
 $\sim$  Dance(Duo{Pair(Girl)}), Dance(Duo{ $\lambda$ })  
 $\sim$  Dance(Duo{Pair(Boy)}), Dance(Duo{ $\lambda$ })  
 Dance(Partaker{Name})  
 $\sim$  Dance(Rating)

Then we repeat the procedure by again searching for an atom  $X$  that satisfies both (a) and (b). After selecting  $Dance(Duo\{Pair(Girl)\})$  we obtain

$\sim$  Dance(Time), Dance(Partaker{Name})  
 $\sim$  Dance(Time), Dance(Duo{Pair(Girl,Boy)})  
 $\sim$  Dance(Time), Dance(Rating)  
 Dance(Duo{Pair(Girl)})  
 Dance(Duo{Pair(Boy)})  
 $\sim$  Dance(Duo{Pair(Boy)}), Dance(Partaker{Name})  
 $\sim$  Dance(Duo{Pair(Girl,Boy)}), Dance(Rating)  
 Dance(Partaker{ $\lambda$ })  
 $\sim$  Dance(Duo{Pair(Girl,Boy)}), Dance(Duo{Pair(Girl)})  
 $\sim$  Dance(Duo{Pair(Girl,Boy)}), Dance(Duo{Pair(Boy)})  
 Dance(Duo{ $\lambda$ })  
 $\sim$  Dance(Duo{Pair(Boy)}), Dance(Duo{ $\lambda$ })  
 Dance(Partaker{Name})  
 $\sim$  Dance(Rating)

and by selecting  $Dance(Duo\{Pair(Boy)\})$  we get

$\sim$  Dance(Time), Dance(Partaker{Name})  
 $\sim$  Dance(Time), Dance(Duo{Pair(Girl,Boy)})  
 $\sim$  Dance(Time), Dance(Rating)  
 Dance(Duo{Pair(Girl)})  
 Dance(Duo{Pair(Boy)})  
 Dance(Partaker{Name})  
 $\sim$  Dance(Duo{Pair(Girl,Boy)}), Dance(Rating)  
 Dance(Partaker{ $\lambda$ })  
 $\sim$  Dance(Duo{Pair(Girl,Boy)}), Dance(Duo{Pair(Girl)})  
 $\sim$  Dance(Duo{Pair(Girl,Boy)}), Dance(Duo{Pair(Boy)})  
 Dance(Duo{ $\lambda$ })  
 Dance(Duo{ $\lambda$ })  
 Dance(Partaker{Name})  
 $\sim$  Dance(Rating)

The entire algorithm halts either when

- 1) the empty string  $\epsilon$  is generated, or when
- 2) there is no atom  $X$  that satisfies both (a) and (b).

If 1) occurs first, that is, the empty string  $\epsilon$  is generated, then  $\sigma$  is a consequence of  $\Sigma$ . If 2) occurs first, then  $\sigma$  is not a consequence of  $\Sigma$ . The algorithm always terminates, and gives a correct answer by Chang's theorem on Horn clauses and our Equivalence Theorem.

In our example the algorithm terminates since no further atom satisfying both (a) and (b) can be found. Therefore,  $\sigma$  is not a consequence of  $\Sigma$ .

## 5 Reusing Relational Design Tools

A direct consequence of Fagin's Equivalence Theorem and Equivalence Theorem 3.1 is that relational database design tools can be applied to solve design problems in complex-value databases. Extended basis attributes of the nested attribute  $N$  are interpreted as flat attributes forming the corresponding relation schema, i.e.,  $R_N = \mathcal{E}(N)$ . Each FD  $\sigma = X \rightarrow Y$  in  $\Sigma$  becomes the relational FD  $\sigma' = \vartheta(X) \rightarrow \vartheta(Y)$ . Given

$\Sigma$  on the nested attribute  $N$ , the corresponding set of relational FDs is

$$\Sigma' = \{\sigma' \mid \sigma \in \Sigma\} \cup \{U \rightarrow V \mid U, V \in \mathcal{E}(N), U \text{ covers } V\}$$

**Corollary 5.1.** *Let  $\Sigma$  be a set of FDs and  $\sigma$  be a single FD, all defined on the nested attribute  $N$ . Then  $\sigma$  is implied by  $\Sigma$  if and only if  $\sigma'$  is implied by  $\Sigma'$  on  $R_N$ .  $\square$*

As a simple example consider again the nested attribute

$$N = \text{Dance}(\text{Time}, \text{Partaker}\{\text{Name}\}, \text{Duo}\{\text{Pair}(\text{Girl}, \text{Boy})\}, \text{Rating}).$$

The extended basis attributes are mapped to flat attribute names as follows:

- Dance(Time) is  $A$ ,
- Dance(Partaker{Name}) is  $B$ ,
- Dance(Partaker{ $\lambda$ }) is  $C$ ,
- Dance(Duo{Pair(Girl,Boy)}) is  $D$ ,
- Dance(Duo{Pair(Girl)}) is  $E$ ,
- Dance(Duo{Pair(Boy)}) is  $F$ ,
- Dance(Duo{ $\lambda$ }) is  $G$ , and
- Dance(Rating) is  $H$ .

The corresponding relation schema is therefore  $R_N = \{A, B, C, D, E, F, G, H\}$ . The FDs in  $\Sigma$  from Subsection 3.1 are encoded as

$$A \rightarrow BDH, B \rightarrow EF, EF \rightarrow B, D \rightarrow G$$

and the structure of  $N$  is encoded by the following FDs:

$$B \rightarrow C, D \rightarrow E, D \rightarrow F, E \rightarrow G, F \rightarrow G \quad .$$

Suppose we want to decide if  $\sigma$  from Subsection 3.1 is a consequence of  $\Sigma$ . Then this is equivalent to deciding whether  $B \rightarrow H$  is a logical consequences of  $\Sigma'$ . In order to decide the last problem, we may apply well-known techniques for relational databases (Beeri & Bernstein 1979) to compute the closure  $B^+ = CEF G$  of  $B$  with respect to  $\Sigma'$ . Since  $H \notin B^+$  the answer to the last problem is no. The Equivalence theorem tells us therefore that  $\sigma$  is not a consequence of  $\Sigma$ . It is future work to exploit the reuse of relational tools further.

## 6 Conclusion and Future Work

We have extended Fagin's well-known equivalence between implications of functional dependencies in relational databases and implications of propositional Horn clauses to functional dependencies in databases that support arbitrary finite nesting using data constructors for records, lists, sets and multisets. Our framework is not based on any specific data model, and the result may therefore lead to a better understanding of complex values in general. The work in (Hartmann et al. 2006) shows that the expressiveness of our functional dependencies is complementary to those investigated in many advanced data models. The extension of the equivalence result follows from a deep case-by-case analysis of the data constructors and the algebraic properties of the database schemata. The results of this article provide two new

ways to look at the implication of functional dependencies in complex-value databases: as the logical implication of a corresponding set of Horn clauses, and as the logical implication of a corresponding set of functional dependencies in relational databases. It is therefore possible to utilise already existing tools from these areas for solving database design problems in the presence of several data constructors.

For the near future we would like to extend the equivalence to multivalued dependencies in the presence of records and lists. It will be challenging to investigate multivalued dependencies in the presence of sets, or to extend results to include (disjoint) unions. There is an alternative proof for the original equivalence result for relational FDs (Fagin 1977). That proof is syntactical in the sense that it takes advantage of the fact that the (finite) implication of FDs can be characterised by a finite, sound and complete set of syntactic inference rules (Armstrong 1974). Such sets of inference rules have also been proposed for the setting of the present paper (Hartmann et al. 2006). A generalisation of these syntactical proofs seems also desirable.

## References

- Abiteboul, S., Hull, R. & Vianu, V. (1995), *Foundations of Databases*, Addison-Wesley.
- Anderson, I. (1987), *Combinatorics of finite sets*, Oxford Science Publications, The Clarendon Press Oxford University Press, New York.
- Armstrong, W. W. (1974), 'Dependency structures of database relationships', *Information Processing* pp. 580–583.
- Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D. & Zdonik, S. (1989), The object-oriented database system manifesto, in 'Proceedings of the International Conference on Deductive and Object-Oriented Databases', pp. 40–57.
- Beeri, C. & Bernstein, P. A. (1979), 'Computational problems related to the design of normal form relational schemata', *Transactions on Database Systems (TODS)* pp. 30–59.
- Biskup, J. (1998), Achievements of relational database schema design theory revisited, in 'Semantics in databases', number 1358 in 'Lecture Notes in Computer Science', Springer, pp. 29–54.
- Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. & Yergeau, F. (2004), 'Extensible markup language (XML) 1.0 (third edition) W3C recommendation 04 february 2004', <http://www.w3.org/TR/2004/REC-xml-20040204/>.
- Chang, C. (1976), Deduce: A deductive query language for relational data bases, in C. H. Chen, ed., 'Pattern Recognition and Artificial Intelligence', Academic Press, New York, pp. 108–134.
- Chang, C. & Lee, R. (1987), *Symbolic Logic and Mechanical Theorem Proving*, Academic Press.
- Chen, P. P. (1976), 'The entity-relationship model: Towards a unified view of data', *Transactions on Database Systems (TODS)* **1**, 9–36.
- Codd, E. F. (1970), 'A relational model of data for large shared data banks', *Communications of the ACM* **13**(6), 377–387.
- Delobel, C. & Adiba, M. (1985), *Relational database systems*, North Holland.
- Delobel, C. & Parker, D. (1978), Functional and multivalued dependencies in a relational database and the theory of Boolean switching functions, Technical report, University of Grenoble, Grenoble, France.
- Dowling, W. & Gallier, J. (1984), 'Linear-time algorithms for testing the satisfiability of propositional horn formulae', *Journal of Logic Programming* **1**(3), 267–284.
- Fagin, R. (1977), 'Functional dependencies in a relational data base and propositional logic', *IBM Journal of Research and Development* **21**(6), 543–544.
- Fagin, R. (1982), 'Horn clauses and data dependencies', *Journal of the ACM* **29**(4), 952–985.
- Hartmann, S., Link, S. & Schewe, K.-D. (2006), 'Axiomatisations of functional dependencies in the presence of records, lists, sets and multisets', accepted for Theoretical Computer Science (TCS).
- Henschen, L. & Wos, L. (1974), 'Unit refutations and horn sets', *Journal of the ACM* **21**(4), 590–605.
- Horn, A. (1951), 'On sentences which are true of direct unions of algebras', *Journal of symbolic logic* **16**, 14–21.
- Hull, R. & King, R. (1987), 'Semantic database modeling: Survey, applications and research issues', *ACM Computing Surveys* **19**(3).
- Li, J., Ng, S. & Wong, L. (2002), Bioinformatics adventures in database research, in 'Proceedings of the International Conference on Database Theory (ICDT)', number 2572 in 'Lecture Notes in Computer Science', Springer, pp. 31–46.
- McKinsey, J. C. C. & Tarski, A. (1946), 'On closed elements in closure algebras', *Annals of Mathematics* **47**, 122–146.
- Parker, D. S. & Delobel, C. (1979), Algorithmic applications for a new result on multivalued dependencies, in 'Proceedings of the International Conference on Very Large Data Bases (VLDB)', pp. 67–74.
- Sagiv, Y. (1980), 'An algorithm for inferring multivalued dependencies with an application to propositional logic', *Journal of the ACM* **27**(2), 250–262.
- Sagiv, Y., Delobel, C., Parker Jr., D. S. & Fagin, R. (1981), 'An equivalence between relational database dependencies and a fragment of propositional logic', *Journal of the ACM* **28**(3), 435–453.
- Suciu, D. (2001), 'On database theory and XML', *SIGMOD Record* **30**(3), 39–45.
- Thalheim, B. (2000), *Entity-Relationship Modeling: Foundations of Database Technology*, Springer-Verlag.
- Vianu, V. (2001), A web odyssey: from Codd to XML, in 'Principles of Database Systems', ACM, pp. 1–15.
- Vincent, M. (1999), 'Semantic foundation of 4NF in relational database design', *Acta Informatica* **36**, 1–41.