Characterization of Database Dependencies with FCA and Pattern Structures

Jaume Baixeries, Mehdi Kaytoue, Amedeo Napoli

Characterization of Database Dependencies with FCA and Pattern Structures

Jaume Baixeries[1], Mehdi Kaytoue[2], and Amedeo Napoli[3]

[1] Universitat Politècnica de Catalunya. 08032, Barcelona. Catalonia.
[2] Université de Lyon. CNRS, INSA-Lyon, LIRIS. UMR5205, F-69621, France.
[3] LORIA (CNRS - Inria Nancy Grand Est - Université de Lorraine), B.P. 239, F-54506, Vandœuvre-lès-Nancy.
jbaixer@lsi.upc.edu,mehdi.kaytoue@insa-lyon.fr,amedeo.napoli@loria.fr

Abstract. In this review paper, we present some recent results on the characterization of Functional Dependencies and variations with the formalism of Pattern Structures and Formal Concept Analysis.
Although these dependencies have been paramount in database theory, they have been used in different fields: artificial intelligence and knowledge discovery, among others.
Keywords: Attribute implications, data dependencies, pattern structures, formal concept analysis, data analysis.

1 Database Dependencies in Data Analysis

In the relational Database Model, a database dependency describes a relationship between sets of attributes that hold in a table. To give a simple and intuitive example of such relations, les us suppose that we have a table with personal details (technically: attributes) of different people: name, age, birth place, average income, and so on. A dependency could state, for instance, that *the age of a person is related to his average income*. This notion of relationship between the attributes of the table is the central idea of a **Database Dependency**. Obviously, this generic idea of relationship may hold different sintactical and semantical formalizations.

However, the idea of data dependency is not only confined to the realm of Database Theory. It can be found in different domains such as Artificial Intelligence, logics or Formal Concept Analysis (FCA), to name a few.

In Artificial Intelligence we may mention, as an example, Decision Trees, which induce a set of relations between the nodes in that tree. In logics, we may mention Horn clauses that formalize the dependency that exist between variables [2]. In FCA we have implications, which explain the dependencies that exist between the attributes in a context [12].

In this paper, we focus on database dependencies that have been defined in the relational database model. In fact, we will focus on functional dependencies (FDs) and derivate dependencies. These are among the most popular types of dependencies [31] since they indicate a functional relation between sets of attributes: the values of a set of attributes are determined by the values of another

set of attributes. To handle errors and uncertainty in real-world data, alternatives exist. *Approximate Dependencies* [15] are FDs that hold in a part –which is user defined– of the database. *Purity Dependencies* [27] express the relationship on the relative impurity induced by two partitions of the table (generated by two sets of attributes). If the impurity is zero, we have a FD.

Another alternative are *Similarity Dependencies* [5], which can be seen as a generalization of Functional Dependencies, but un-crispring the basic definition of FDs: similar values of an attribute determine similar values of another attribute. Similarity has been considered for FDs under several terms, e.g. fuzzy FDs [8], matching dependencies [29], constraint generating dependencies [7]. Moreover, it is still an active topic of research in the database community [9,10,29,30].

Because of the interest that those dependencies have in Data Analysis, in this paper we want to address their characterization with the formalism of Formal Concept Analysis [12]. The choice of this formalism is explained by the important research in this field that has been devoted to the computation of implications, functional dependencies, multivalued dependencies, and other kinds of dependencies [3,1,21,23,22,4,5,8,34]. Formal Concept Analysis has been proved, as well, as a valid formalism for different fields of Knowledge Discovery [24,25,20], Machine Learning [18] or gene mining [16,17], among others.

This paper is organized as follows: first, we present the formal definitions for the different kinds of dependencies that will be discussed, focusing in detail in Functional and Similarity Dependencies. Next, we present some computational issues of those dependencies within the classical FCA framework, and, finally, we present an alternative framework. The main goal of this paper is to present recent advancements in the characterization of functional and similarity dependencies, so that they can be easily used in the process of analyzing large datasets.

2 Functional Dependencies and Variations

2.1 Notation

We deal with datasets which are sets of tuples. Let $\mathcal{U}$ be a set of attributes and *Dom* be a set of values (a domain). For the sake of simplicity, we assume that *Dom* is a numerical set. A tuple t is a function $t : \mathcal{U} \mapsto Dom$ and then a table T is a set of tuples. Usually a table is presented as a matrix, as in the table of Example 1, where the set of tuples (or objects) is $T = \{t_1, t_2, t_3, t_4\}$ and $\mathcal{U} = \{a, b, c, d\}$ is the set of attributes.

The functional notation allows one to associate an attribute with its value. We define the functional notation of a tuple for a set of attributes X as follows, assuming that there exists a total ordering on $\mathcal{U}$. Given a tuple $t \in T$ and $X = \{x_1, x_2, \ldots, x_n\} \subseteq \mathcal{U}$, we have:

$$t(X) = \langle t(x_1), t(x_2), \ldots, t(x_n) \rangle$$

In Example 1, we have $t_2(\{a, c\}) = \langle t_2(a), t_2(c) \rangle = \langle 4, 4 \rangle$. In this paper, the set notation is usually omitted and we write ab instead of $\{a, b\}$.

Example 1. This is an example of a table $T = \{t_1, t_2, t_3, t_4\}$, based on the set of attributes $\mathcal{U} = \{a, b, c, d\}$.

id	a	b	c	d
t_1	1	3	4	1
t_2	4	3	4	3
t_3	1	8	4	1
t_4	4	3	7	3

We are also dealing with the set of partitions of a set. Let S be any arbitrary finite set, then $\text{Part}(S)$ is the set of all possible partitions that can be formed with S. The set of partitions of a set is a lattice [13]. We recall that partitions can also be considered as equivalence classes induced by an equivalence relation.

2.2 Functional Dependencies

We now introduce functional dependencies (FDs).

Definition 1 ([31]). *Let T be a set of tuples (or a data table), and $X, Y \subseteq \mathcal{U}$. A* functional dependency (FD) *$X \to Y$ holds in T if*

$$\forall t, t' \in T : t(X) = t'(X) \Rightarrow t(Y) = t'(Y)$$

For example, the functional dependencies $a \to d$ and $d \to a$ hold in the table of Example 1, whereas the functional dependency $a \to c$ does not hold since $t_2(a) = t_4(a)$ but $t_2(c) \neq t_4(c)$.

There is an alternative way of considering Functional Dependencies using partitions of the set of tuples T. Taking a set of attributes $X \subseteq \mathcal{U}$, we define the partition of tuples induced by this set as follows.

Definition 2. *Let $X \subseteq \mathcal{U}$ be a set of attributes in a table T. Two tuples t_i and t_j in T are equivalent w.r.t. X when:*

$$t_i \sim t_j \iff t_i(X) = t_j(X)$$

Then, the partition of T induced by X is a set of equivalence classes:

$$\Pi_X(T) = \{c_1, c_2, \ldots, c_m\}$$

For example, if we consider the table in Example 1, we have $\Pi_a(T) = \{\{t_1, t_3\}, \{t_2, t_4\}\}$.

The set of all partitions of a set T is $\text{Part}(T)$. We can also notice that the set of partitions of any set $\text{Part}(T)$ induces an ordering relation $\leq$:

$$\forall P_i, P_j \in \text{Part}(T) : P_i \leq P_j \iff \forall c \in P_i : \exists c' \in P_j : c \subseteq c'$$

For example: $\{\{t_1\}, \{t_2\}, \{t_3, t_4\}\} \leq \{\{t_1\}, \{t_2, t_3, t_4\}\}$. According to the partitions induced by a set of attributes, we have an alternative way of defining necessary and sufficient conditions for a functional dependency to hold:

Proposition 1 ([15]). *A functional dependency $X \to Y$ holds in T if and only if $\Pi_Y(T) \leq \Pi_X(T)$.*

Again, taking the table in Example 1, we have that $a \to d$ holds and that $\Pi_d \leq \Pi_a$ since $\Pi_a(T) = \{\{t_1, t_3\}, \{t_2, t_4\}\}$ and $\Pi_d(T) = \{\{t_1, t_3\}, \{t_2, t_4\}\}$ (actually $d \to a$ holds too).

2.3 Purity and Approximate Dependencies

We now present the definition of two generalizations of Functional Dependencies: Purity Dependencies and Approximate Dependencies.

Approximate Dependencies [15]. In a table, there may be some tuples that prevent a functional dependency from holding. Those tuples can be seen as exceptions (or errors) for that dependency. Removing such tuples allows the dependency to exist: then a threshold can be set to define a set of "approximate dependencies" holding in a table. For example, a threshold of 10% means that all functional dependencies holding after removing up to 10% of the tuples of a table are valid approximate dependencies. The set of tuples to be removed for validating a functional dependency does not need to be the same for each approximate dependency. Considering the dependency in Example 2 $Month \to Av.Temp$, we can check that 6 tuples should be removed before verifying the dependency: we keep only one tuple for `Month 1` and one tuple for `Month 5` (actually just as if we remove "duplicates"). Then, if the threshold is equal to or larger than 75%, $Month \to Av.Temp$ is a valid Approximate Dependency.

Example 2. This table is an excerpt of the *Average Daily Temperature Archive* [4] from The University of Dayton, that shows the month average temperatures for different cities.

id	Month	Year	Av. Temp.	City
t_1	1	1995	36.4	Milan
t_2	1	1996	33.8	Milan
t_3	5	1996	63.1	Rome
t_4	5	1997	59.6	Rome
t_5	1	1998	41.4	Dallas
t_6	1	1999	46.8	Dallas
t_7	5	1996	84.5	Houston
t_8	5	1998	80.2	Houston

Purity Dependencies [27] are a generalization of the relationship between partitions induced by the left-hand side and right-hand side of a functional dependency. These dependencies are based on the relative impurity measure of two partitions. In order to compute this impurity measure, we need a concave and subadditive function defined on the interval $[0, 1]$ (for example, the binary entropy function). The intuition about this measure is that it computes *how much those partitions disagree*, i.e. how far two partitions π and σ are from fulfilling the relation $\pi \leq \sigma$. In the case of approximate dependencies, this measure was the number (or percentage) of tuples that had to be removed, but as for purity dependencies, this measure can be more general. If the impurity measure is zero (or close to zero), then $\pi \leq \sigma$. It indicates that those tuples fulfil (or almost fulfil) the relation $\pi \leq \sigma$. The *closer* they are to the relation $\pi \leq \sigma$, the closer to zero will be their relative impurity.

For example, the impurity measure (details on this measure are given in [26]) of partition $\{\{1, 2, 3\}, \{4, 5\}\}$ w.r.t. partition $\{\{1, 2\}, \{3, 4, 5\}\}$ is 5.6, whereas the impurity measure of partition $\{\{1, 3\}, \{2, 5\}, \{4\}\}$ w.r.t. partition $\{\{1, 2\}, \{3, 4, 5\}\}$ is 8.2. In the first pair of partitions, only tuple 3 is misplaced, i.e. moving 3 from

one partition to another leads to the the same partitions, whereas in the second example, the number of misplaced elements is larger (2, 3, and 4 should be moved).

An important feature of this measure is that if a partition is finer than another, then, their relative impurity measure is exactly 0. This implies that a purity dependency $X \rightarrow Y$ holds if and only if the relative impurity of $\Pi_X(T)$ w.r.t. $\Pi_Y(T)$ is below a user-defined threshold. Therefore, if $\Pi_Y(T) \leq \Pi_X(T)$, a functional dependency is a valid purity dependency, regardless of the threshold.

2.4 Similarity Dependencies

Similarity Dependencies [5] (these dependencies are called Differential Dependencies in [28]) are another generalization of Funcional Dependencies. The intuition behind these dependencies is that in Similarity Dependencies, we relax the condition that states that $X \rightarrow Y$ holds if and only if for each pair of tuples, the fact that if their values in attributes X are the same, their values in Y are the same as well. For similarity dependencies, we change the *are the same* by *are similar*.

In order to define Similarity Dependencies, we first define a *tolerance* relation in a set S:

Definition 3. *$\theta \subseteq S \times S$ is a tolerance relation if:*

1. $\forall s_i \in S : s_i \theta s_i$ *(reflexivity)*
2. $\forall s_i, s_j \in S : s_i \theta s_j \iff s_j \theta s_i$ *(symmetry)*

A tolerance relation is not necessarily transitive. Precisely, in this detail lies the difference between Functional and Similarity Dependencies: in FD's we had that the relation $=$ induced an equivalence relation, because it is obviously transitive, but in this case, we do not need to enforce this condition.

An example of a tolerance relation is the *similarity* that can be defined within a set of integer values as follows. Given two integer values v_1, v_2 and a threshold ϵ (user-defined): $v_1 \theta v_2 \iff |v_1 - v_2| \leq \epsilon$. For example, when $S = \{1, 2, 3, 4, 5\}$ and $\epsilon = 2$, then $S/\theta = \{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}\}$. S/θ is not a partition as transitivity does not apply.

A tolerance relation induces *blocks of tolerance*:

Definition 4. *Given a set S, a subset $K \subseteq S$, and a tolerance relation $\theta \subseteq S \times S$, K is a* block of tolerance *of θ if:*

1. $\forall x, y \in K : x \theta y$ *(pairwise correspondence)*
2. $\forall z \notin K, \exists u \in K : \neg(z \theta u)$ *(maximality)*

All elements in a tolerance block are in pairwise correspondence, and the block is maximal with respect to the relation θ. The set of all tolerance blocks induced by a tolerance relation θ on the set S is denoted by S/θ (by analogy with the notation of equivalence classes). S/θ is a set of maximal subsets of S and as

such, $S/\theta \in \wp(\wp(S))$. The set of sets of tolerance blocks also induces a partial order, as in the case of partitions. In fact, a tolerance relation and tolerance blocks are a generalization of equivalence relations and equivalence classes.

Two tuples are similar w.r.t. a set of attributes X if and only if they are similar w.r.t. *each* attributes in X. We now can define a *similarity dependency*:

Definition 5. *Let $X, Y \subseteq \mathcal{U}$: $X \rightarrow Y$ is a similarity dependency iff $\forall t_i, t_j \in T$: $t_i \theta_X t_j \Rightarrow t_i \theta_Y t_j$*

Example 3. Going back to example 2, let us compute the Similarity Dependencies that hold and that have the attribute *Av. Temp.* in their right-hand side).

Dependency	Holds
`Month -> Av. Temp`	N
`Month, Year -> Av. Temp`	N
`Month, City -> Av. Temp`	Y
`Year -> Av. Temp`	N
`Year, City -> Av. Temp`	N
`City -> Av. Temp`	N

The only similarity dependency that holds is $Month, City \rightarrow Av.Temp$, using the following similarity measures for each attribute: $x\ \theta_{Month}\ y \iff |x - y| \leq 0$, $x\ \theta_{Year}\ y \iff |x - y| \leq 0$, $x\ \theta_{City}\ y \iff distance(x, y) \leq 500$ and $x\ \theta_{Av.Temp}\ y \iff |x - y| \leq 10$.

The similarity imposes that the month and year must be the same, whereas the distance between cities should be less than 500 kilometers and the difference between average temperatures should be less than 10 degrees (all these values are of course arbitrary).

In particular, considering the tuples t_1, t_2: $t_1 \theta_{Month,City} t_2$ since $t_1(Month) = t_2(Month) = \langle\, 1 \,\rangle$ and $t_1(City) = t_2(City) = \langle\, Milan \,\rangle$. From the other side, we have that $t_1 \theta_{Av.Temp.} t_2$ since $|36.4 - 33.8| \leq 10$.

3 Formal Concept Analysis

Formal Concept Analysis (FCA) [12] is a mathematical framework allowing to build a concept lattice from a binary relation between objects and their attributes. The concept lattice can be represented by a diagram where classes of objects/attributes and ordering relations between classes can be drawn, interpreted and used for data-mining, knowledge management and discovery [32,33].

We use standard definitions from [12]. Let G and M be arbitrary sets and $I \subseteq G \times M$ be an arbitrary binary relation between G and M. The triple (G, M, I) is called a formal context. Each $g \in G$ is interpreted as an object, each $m \in M$ is interpreted as an attribute. The statement $(g, m) \in I$ is interpreted as "g has attribute m". The two following derivation operators $(\cdot)'$:

$$A' = \{m \in M \mid \forall g \in A : gIm\} \qquad for\ A \subseteq G,$$
$$B' = \{g \in G \mid \forall m \in B : gIm\} \qquad for\ B \subseteq M$$

define a Galois connection between the powersets of G and M. The derivation operators $\{(\cdot)', (\cdot)'\}$ put in relation elements of the lattices $(\wp(G), \subseteq)$ of objects and $(\wp(M), \subseteq)$ of attributes and reciprocally. A Galois connection induces closure operators $(\cdot)''$ and realizes a one-to-one correspondence between all closed sets of objects and all closed sets of attributes. For $A \subseteq G$, $B \subseteq M$, a pair (A, B) such that $A' = B$ and $B' = A$, is called a *formal concept*. Concepts are partially ordered by $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2$ ($\Leftrightarrow B_2 \subseteq B_1$). (A_1, B_1) is a sub-concept of (A_2, B_2), while the latter is a super-concept of (A_1, B_1). With respect to this partial order, the set of all formal concepts forms a complete lattice called the *concept lattice* of the formal context (G, M, I), i.e. any subset of concepts has both a supremum (join $\vee$) and an infimum (meet $\wedge$) [12]. For a concept (A, B) the set A is called the *extent* and the set B the *intent* of the concept. The set of all concepts of a formal context (G, M, I) is denoted by $\mathfrak{B}(G, M, I)$ while the concept lattice is denoted by $\underline{\mathfrak{B}}(G, M, I)$.

An implication of a formal context (G, M, I) is denoted by $X \to Y$, $X, Y \subseteq M$ and means that all objects from G having the attributes in X also have the attributes in Y, i.e. $X' \subseteq Y'$. Implications obey the Amstrong rules (reflexivity, augmentation, transitivity). A minimal subset of implications (in sense of its cardinality) from which all implications can be deduced with Amstrong rules is called the Duquenne-Guigues basis [14].

Objects described by non binary attributes can be represented in FCA as a many-valued context (G, M, W, I) with a set of objects G, a set of attributes M, a set of attribute values W and a ternary relation $I \subseteq G \times M \times W$. The statement $(g, m, w) \in I$, also written $g(m) = w$, means that "the value of attribute m taken by object g is w". The relation I verifies that $g(m) = w$ and $g(m) = v$ always implies $w = v$. For applying the FCA machinery, a many-valued context can be transformed into a formal context with a conceptual scaling. The choice of a scale should be wisely done w.r.t. data and goals since it affects the size, the interpretation, and the computation of the resulting concept lattice. This will be discussed in the next section.

4 Computation of FD's

There are different algorithms that deal with the computation of FD's. One of the most well-known is TANE [15], which is based on the incremental computation of equivalence classes of tuples. FD Mine [35] takes the approach of reducing the number of candidate dependencies by using the axioms that apply to them. Approximate Dependencies computation is explained in [15], and other different kind of FD-alike dependencies can be found in [29] and [30].

In this paper, we focus on the computation of FD's and Similarity Dependencies within the framework of Formal Concept Analysis. The most well-known method [12,1] is called *binarization*, and consists in transforming (implicitly) a many-valued set of data into a binary context. This transformation allows us to build a formal context. In order to define this formal context, we need to define first the set of objects:

$$G = \{(t_i, t_j) \mid i < j \text{ and } t_i, t_j \in T\}$$

This corresponds to the set of all pairs of tuples from T (excluding symmetry and reflexivity). The relation of the context is defined as:

$$(t_i, t_j)\ I\ x \Leftrightarrow t_i(x) = t_j(x)$$

It is important to realize that the formal context $\mathbb{K} = (G, \mathcal{U}, I)$ depends entirely on the table T, since both G and $\mathcal{U}$ do, but this dependency is not explicitly shown in the definition of this context. Figure 1 presents an example of this binarization.

We can see that the size of this context can be of the order of $\mathcal{O}(|T^2|)$ (where $|T|$ is the number of tuples of T), so it can be significantly bigger than the original set of data.

id	a	b	c	d
t_1	1	2	3	1
t_2	1	2	1	4
t_3	1	1	3	4
t_4	2	2	3	4

id	a	b	c	d
(t_1, t_2)	x	x		
(t_1, t_3)	x		x	
(t_1, t_4)		x	x	
(t_2, t_3)	x			x
(t_2, t_4)		x		x
(t_3, t_4)			x	x

Fig. 1. A data table T (left) with its associated formal context $(\mathcal{B}_2(G), M, I)$ (right).

Another FCA-oriented option is that of **scaling**. This alternative consists in creating a formal context such that its attributes are all possible values that can appear in a table (see [12] for a more detailed explanation). The problem of this approach is that it depends on the number of different values that a table may contain, which, in most cases, can be an extreamly large number of them.

Both methods suffer from the same drawback: the fact that, the transformation of the original data into a formal context can be too costly in terms of size w.r.t. the original dataset.

In order to overcome this burden, [4] and [5] propose the use of Pattern Structures (an extension of Formal Concept Analysis). In the next section, we review some details of this approach.

5 Pattern Structures in Formal Concept Analysis

Our interest lies in handling numerical data within FCA. Hence, we recall here the formalism of pattern structures that can be understood as a generalization towards complex data, i.e. objects taking descriptions in a partially ordered set. In the case of pattern structures, the descriptions will be taken from meet-semi-lattices, which arise naturally from a partial order [19].

A pattern structure is defined as a generalization of a formal context describing complex data [11]. Formally, let G be a set of objects, let $(D, \sqcap)$ be a meet-semi-lattice of potential object descriptions and let $\delta : G \longrightarrow D$ be a mapping associating each object with its description. Then $(G, (D, \sqcap), \delta)$ is a pattern structure. Elements of D are patterns and are ordered by a subsumption relation $\sqsubseteq$: $\forall c, d \in D,\ c \sqsubseteq d \Longleftrightarrow c \sqcap d = c$. A pattern structure $(G, (D, \sqcap), \delta)$ gives rise to two derivation operators $(\cdot)^{\square}$:

$$A^{\square} = \prod_{g \in A} \delta(g) \qquad for\ A \subseteq G$$

$$d^{\square} = \{g \in G | d \sqsubseteq \delta(g)\} \qquad for\ d \in (D, \sqcap).$$

These operators form a Galois connection between $(2^G, \subseteq)$ and $(D, \sqcap)$. Pattern concepts of $(G, (D, \sqcap), \delta)$ are pairs of the form (A, d), $A \subseteq G$, $d \in (D, \sqcap)$, such that $A^{\square} = d$ and $A = d^{\square}$. For a pattern concept (A, d), d is a pattern intent and is the common description of all objects in A, the pattern extent. When partially ordered by $(A_1, d_1) \leq (A_2, d_2) \Leftrightarrow A_1 \subseteq A_2\ \ (\Leftrightarrow d_2 \sqsubseteq d_1)$, the set of all concepts forms a complete lattice called pattern concept lattice.

6 Functional Dependencies and Their Variations with FCA and Pattern Structures

Consider a numerical table as a many-valued context (G, M, W, I) where G corresponds to the set of objects ("rows"), M to the set of attributes ("columns"), W the data domain ("all distinct values of the table") and $I \subseteq G \times M \times W$ a relation such that $(g, m, w) \in I$ also written $m(g) = w$ means that attribute m takes the value w for the object g [12].

We now show how the Functional Dependencies and Similarity Dependencies can be characterized using pattern structures and FCA. We first have to define the set of formal objects, which in both cases are the set of attributes that are present in the original table. Then, given an attribute $m \in M$, its description $\delta(m)$ is given by the sets induced by the respective relations within the set of tuples: the equivalence relation defined in Section 2.2 for Functional Dependencies and the tolerance relation defined in Section 2.4 for Similarity Dependencies. In both cases, the description of an attribute is a set of sets of tuples. In the case of FD's, this will be a partition, in the case of Similarity Dependencies, it will be a set of tolerance blocks. For instance, in the case of Functional Dependencies, the description of an attribute will be given by a partition over G such that any two elements g, h of the same class take the same values for the attribute m, i.e. $m(g) = m(h)$.

Since in both cases the set of descriptions obey to a partial order, our initial numerical table (G, M, W, I) can be represented as a pattern structure $(M, (D, \sqcap, \sqcup), \delta)$ where M is the set of original attributes, and $(D, \sqcap, \sqcup)$ is the lattice of partitions or tolerance blocks over G.

Therefore, we have that, for a given set of attributes $X \subseteq \mathcal{U}$, its description is $\{X\}^{\square}$. We remark that $\{X\}^{\square}$ depends on the definition of the pattern formal

context defined, which, in turn, depends on the definition of the description of the attributes. Table 1 shows an example of the description of all the attributes of a table, in the case that the description is computed according to the binary relation in Definition 2.

id	A	B	C	D
1	1	3	7	2
2	1	3	4	5
3	3	5	2	2
4	3	3	4	8

$m \in M$	$\delta(m) \in (D, \sqcap, \sqcup)$
A	$\{\{1,2\},\{3,4\}\}$
B	$\{\{1,2,4\},\{3\}\}$
C	$\{\{1\},\{2,4\},\{3\}\}$
D	$\{\{1,3\},\{2\},\{4\}\}$

Table 1. The original data (left), the resulting pattern structure (right)

We now can state how this formal pattern context characterizes the set of Functional Dependencies or Similarity Dependencies that hold in a dataset:

Proposition 2 ([5]). *A functional (similarity) dependency $X \to Y$ holds in a table T if and only if: $\{X\}^{\square} = \{XY\}^{\square}$ in the pattern structure $(M, (D, \sqcap), \delta)$.*

7 Conclusions and Future Work

In this paper we have presented Functional Dependencies and other generalizations, and we have discussed how those dependencies are relevant and of interest to data analysis, and we have focused on FCA-based characterizations.

We have also discussed that the classical methods in FCA for computing dependencies have a computational cost that may be, in some cases, unfeasable. Pattern Structures are a way to overcome this problem. We have presented a way to apply this framework in order to characterize Functional and Similarity Dependencies. Experiments [5,6] seem to confirm the validity of this approach.

Future work should advance into two different (yet, complementary) paths: on the one hand, it is needed to perform more experiments and evaluate them more precisely, in terms of speed as well as memory usage. On the other hand, it is also needed to use this same framework in order to characterize other kinds of dependencies which have been discussed in this paper. This would include Approximate, Purity, Fuzzy Dependencies, among others.

Acknowledgments. This research work has been supported by the Spanish Ministry of Education and Science (project TIN2008-06582-C03-01), EU PASCAL2 Network of Excellence, and by the Generalitat de Catalunya (2009-SGR-980 and 2009-SGR-1428) and AGAUR (grant 2010PIV00057).

References

1. J. Baixeries. *Lattice Characterization of Armstrong and Symmetric Dependencies (PhD Thesis).* Universitat Politècnica de Catalunya, 2007.

2. J. Baixeries and J. L. Balcázar. Discrete deterministic data mining as knowledge compilation. In *Proceedings of Workshop on Discrete Mathematics and Data Mining – SIAM*, 2003.
3. J. Baixeries and J. L. Balcázar. A lattice representation of relations, multivalued dependencies and armstrong relations. In *ICCS*, pages 13–26, 2005.
4. J. Baixeries, M. Kaytoue, and A. Napoli. Computing functional dependencies with pattern structures. In L. Szathmary and U. Priss, editors, *CLA*, volume 972 of *CEUR Workshop Proceedings*, pages 175–186. CEUR-WS.org, 2012.
5. J. Baixeries, M. Kaytoue, and A. Napoli. Computing similarity dependencies with pattern structures. In *CLA*, pages 33–44, 2013.
6. J. Baixeries, M. Kaytoue, and A. Napoli. Characterizing functional dependencies in formal concept analysis with pattern structures. *Annals of Mathematics and Artificial Intelligence*, pages 1–21, 2014.
7. M. Baudinet, J. Chomicki, and P. Wolper. Constraint-generating dependencies. *J. Comput. Syst. Sci.*, 59(1):94–115, 1999.
8. R. Belohlávek and V. Vychodil. Data tables with similarity relations: Functional dependencies, complete rules and non-redundant bases. In M.-L. Lee, K.-L. Tan, and V. Wuwongse, editors, *DASFAA*, volume 3882 of *Lecture Notes in Computer Science*, pages 644–658. Springer, 2006.
9. L. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *Proceedings of the 14th International Conference on Database Theory*, ICDT '11, pages 268–279, New York, NY, USA, 2011. ACM.
10. W. Fan, H. Gao, X. Jia, J. Li, and S. Ma. Dynamic constraints for record matching. *The VLDB Journal*, 20(4):495–520, Aug. 2011.
11. B. Ganter and S. O. Kuznetsov. Pattern structures and their projections. In H. S. Delugach and G. Stumme, editors, *Conceptual Structures: Broadening the Base, Proceedings of the 9th International Conference on Conceptual Structures (ICCS 2001)*, LNCS 2120, pages 129–142. Springer, 2001.
12. B. Ganter and R. Wille. *Formal Concept Analysis.* Springer, Berlin, 1999.
13. G. Graetzer, B. Davey, R. Freese, B. Ganter, M. Greferath, P. Jipsen, H. Priestley, H. Rose, E. Schmidt, S. Schmidt, F. Wehrung, and R. Wille. *General Lattice Theory.* Freeman, San Francisco, CA, 1971.
14. J.-L. Guigues and V. Duquenne. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines*, 95:5–18, 1986.
15. Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *Computer Journal*, 42(2):100–111, 1999.
16. M. Kaytoue, S. O. Kuznetsov, and A. Napoli. Revisiting numerical pattern mining with formal concept analysis. In *IJCAI*, pages 1342–1347, 2011.
17. M. Kaytoue, S. O. Kuznetsov, A. Napoli, and S. Duplessis. Mining Gene Expression Data with Pattern Structures in Formal Concept Analysis. *Information Science*, 181(10):1989–2001, 2011.
18. S. Kuznetsov. Mathematical aspects of concept analysis. *Journal of Mathematical Sciences*, 80(2):1654–1698, 1996.
19. S. O. Kuznetsov. Fitting pattern structures to knowledge discovery in big data. In P. Cellier, F. Distel, and B. Ganter, editors, *ICFCA*, volume 7880 of *Lecture Notes in Computer Science*, pages 254–266. Springer, 2013.

20. S. O. Kuznetsov and J. Poelmans. Knowledge representation and processing with formal concept analysis. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 3(3):200–215, 2013.
21. S. Lopes, J.-M. Petit, and L. Lakhal. Functional and approximate dependency mining: database and fca points of view. *Journal of Experimental and Theoretical Artificial Intelligence*, 14(2-3):93–114, 2002.
22. R. Medina and L. Nourine. A unified hierarchy for functional dependencies, conditional functional dependencies and association rules. In S. Ferré and S. Rudolph, editors, *ICFCA*, volume 5548 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2009.
23. S. Nedjar, F. Pesci, L. Lakhal, and R. Cicchetti. The agree concept lattice for multidimensional database analysis. In P. Valtchev and R. Jäschke, editors, *ICFCA*, volume 6628 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2011.
24. J. Poelmans, D. I. Ignatov, S. O. Kuznetsov, and G. Dedene. Formal concept analysis in knowledge processing: A survey on applications. *Expert Syst. Appl.*, 40(16):6538–6560, 2013.
25. J. Poelmans, S. O. Kuznetsov, D. I. Ignatov, and G. Dedene. Formal concept analysis in knowledge processing: A survey on models and techniques. *Expert Syst. Appl.*, 40(16):6601–6623, 2013.
26. D. Simovici and S. Jaroszewicz. An axiomatization of partition entropy. *Information Theory, IEEE Transactions on*, 48(7):2138–2142, 2002.
27. D. A. Simovici, D. Cristofor, and L. Cristofor. Impurity measures in databases. *Acta Inf.*, 38(5):307–324, 2002.
28. S. Song and L. Chen. Differential dependencies: Reasoning and discovery. *ACM Trans. Database Syst.*, 36(3):16:1–16:41, Aug. 2011.
29. S. Song and L. Chen. Efficient discovery of similarity constraints for matching dependencies. *Data & Knowledge Engineering*, 2013. (in press).
30. S. Song, L. Chen, and P. S. Yu. Comparable dependencies over heterogeneous data. *The VLDB Journal*, 22(2):253–274, Apr. 2013.
31. J. Ullman. *Principles of Database Systems and Knowledge-Based Systems, volumes 1–2*. Computer Science Press, Rockville (MD), USA, 1989.
32. P. Valtchev, R. Missaoui, and R. Godin. Formal concept analysis for knowledge discovery and data mining: The new challenges. In P. W. Eklund, editor, *ICFCA*, volume 2961 of *Lecture Notes in Computer Science*, pages 352–371. Springer, 2004.
33. R. Wille. Why can concept lattices support knowledge discovery in databases? *Journal of Experimental and Theoretical Artificial Intelligence*, 14(2-3):81–92, 2002.
34. C. Wyss, C. Giannella, and E. L. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract. In *Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery*, DaWaK '01, pages 101–110, London, UK, UK, 2001. Springer-Verlag.
35. H. Yao and H. J. Hamilton. Mining functional dependencies from data. *Data Min. Knowl. Discov.*, 16(2):197–219, Apr. 2008.