Computing Dependencies Using Formal Concept Analysis

Jaume Baixeries, Victor Codocedo, Mehdi Kaytoue, Amedeo Napoli

Chapter 6
Computing Dependencies Using FCA

Jaume Baixeries, Victor Codocedo, Mehdi Kaytoue, and Amedeo Napoli

6.1 Introduction

The goal of this paper is to present and discuss some of our recent advances on the characterization and computation of different database constraints with Formal Concept Analysis [24] enriched with the formalism of pattern structures [23].

Although the Relational Database Model (RDBM) [31] currently coexists with other non-SQL database models [22], it still is one of the most popular database systems nowadays. In this model, one of the most important constraints is based on **Functional Dependencies** [30]. These dependencies are of key importance in the normalization of a database schema (the process of splitting an original dataset into smaller units in order to prevent redundancies and anomalies in the update process), and also in some accessory tasks, like data cleaning [15].

However, the definition of FDs is too strict for several useful tasks that are not related to the design of a database. One prototypical example is when one should model datasets with imprecision. By this, we mean datasets that contain errors and uncertainty in real-world data. To overcome this problem, different generalizations of FDs have been defined. These generalizations can be classified according to the criteria by which they relax the equality condition of FDs [16]. According to this

J. Baixeries (✉)
Computer Science Department, Universitat Politècnica de Catalunya, Barcelona, Spain
e-mail: jbaixer@cs.upc.edu

V. Codocedo
Departamento de Informática, Universidad Técnica Federico Santa María, Campus San Joaquín, Santiago de Chile, Chile
e-mail: victor.codocedo@inf.utfsm.cl

M. Kaytoue
Infologic R&D, Bourg-Lès-Valence, France
e-mail: mehdi.kaytoue@insa-lyon.fr

A. Napoli
Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
e-mail: amedeo.napoli@loria.fr

R. Missaoui et al. (eds.), *Complex Data Analytics with Formal Concept Analysis*,
https://doi.org/10.1007/978-3-030-93278-7_6

classification, two main strategies are presented: "extent relaxation" and "attribute relaxation" (in agreement with the terminology introduced in [16]).

The difference between these two strategies is that in the case of *extent relaxation*, a functional dependency may hold *only* in a (predefined) fraction of the datasets. In the *attribute relaxation* case, the equality that holds in the definition of a functional dependency may be *relaxed*, this is, it can be generalized by a more relaxed and general definition of an equality. The dependencies that follow this latter strategy have been named in different ways: fuzzy FDs [12], matching dependencies [33], constraint generating dependencies [13]. We name them **Similarity Dependencies**. These dependencies are still an active topic of research in the database community [14, 21, 33, 34].

Formal Concept Analysis (FCA) is a branch of applied lattice theory that dates back to the 1980s [24]. It is based on a binary relation between a set of objects and a set of attributes, and constructs a set of formal concepts with two operators between objects and attributes that form a Galois connection. Formal Concept Analysis has proved to be useful in many different fields: artificial intelligence, knowledge management, data-mining and machine learning, morphological mathematics, etc. [27].

Pattern structures [28] are a generalization of FCA for dealing with complex data within the framework of FCA [26, 28]. One of the interesting features of pattern structures is that they do not need to transform the original dataset before any treatment. As we will later see, this is one keypoint of the results presented here.

The characterization of FDs is a fruitful field in lattice theory [17, 20, 29]. Since the characterization of FDs with Formal Concept Analysis (FCA) was originally presented in [24], many papers have continued this line of research. Although this list is not intended to be exhaustive, we name the following general lines of research:

- Characterization of Functional Dependencies [2, 29].
- Characterization of Similarity Dependencies [12, 29].
- Characterization of other Database Dependencies [4–6, 9, 32].

In this paper, we present some recent results that deal with the characterization of FDs and similarity dependencies in terms of FCA and Pattern Structures, which have already been published [8–11]. The aim of this paper is not only to offer a general overview of the results obtained by these authors, but to try to present a more general overview of this topic, as well as a discussion of the limitations and potential possibilities that this line of research offers to both researchers and practitioners.

This line of research is relevant to different fields of research, namely, the database dependencies computation in the relational model, since we provide the possibility that existing algorithms in FCA may be used in the relational database model, or complex database analysis, since the generalization of functional dependencies allows the modeling of more complex relations within a dataset.

We present first the basic notations in Sect. 6.2, and then, we present some previous results that link FCA with the characterization of FDs. We introduce the results in Sect. 6.4, which are later discussed in Sect. 6.5.

6.2 Notation

In this section, we present the notation that will be used in this paper. The main object that will be used is a **dataset** D (equivalently **table, dataset, set of tuples**), that is composed by a set of **attributes** $\mathscr{U}$ and a set of **tuples** T, this is: $D = (\mathscr{U}, T)$. Along the paper, we may refer to either a dataset $D = (\mathscr{U}, T)$ or to the set of tuples T that it contains, in which case, we assume also the set of attributes $\mathscr{U}$.

The domain of $\mathscr{U}$ is Dom, which is a set of values. In the forthcoming examples, we assume that Dom is a numerical set. A tuple t is a function $t : \mathscr{U} \mapsto Dom$. Usually tables are presented as in Table 6.1, where the set of tuples (or objects) is $T = \{t_1, t_2, t_3, t_4\}$, $\mathscr{U} = \{a, b, c, d\}$ is the set of attributes, and $Dom = \{1, 3, 4, 7, 8\}$. Given a tuple $t \in T$, we say that $t(X)$ (for all $X \subseteq \mathscr{U}$) is a tuple with the values of t in the attributes $x_i \in X$:

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

For example, we have that $t_2(\{a, c\}) = \langle t_2(a), t_2(c) \rangle = \langle 4, 4 \rangle$. We also drop the set notation: instead of $\{a, b\}$ we use ab.

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

Table 6.1: An example of a table D, i.e. a set of tuples

6.2.1 Equivalence Relation

An equivalence relation R on a set T can be seen as:

1. A set of pairs, since $R \subseteq T \times T$.
2. A partition of the set T, this is, a set of subsets of $P = \{S_1, S_2, \ldots, S_n\}$, such that each pair $S_i \cap S_j = \emptyset$ when $i \neq j$ and $T = S_1 \cup S_2 \cup \cdots \cup S_n$. Each S_i is a **class** of T.

For example, given $P = \{\{1, 2, 3\}, \{4\}\}$, one has the relation

$$R_P = \{(1,2), (1,3), (2,3), (1,1), (2,2), (3,3), (4,4)\}$$

(omitting symmetry for the sake of readability). Equivalence relations contain also a partial order with the set inclusion operator when the equivalence relation is seen as a set of pairs, where the **meet** and **join** of two equivalence relations are the intersection and the union respectively. If this relation is seen as partitions of a set, the order if

defined by the **finer** (**coarser**) relation: a partition P_i is finer than a partition P_j if every class of P_i is a subset of a class of P_j. The coarser relation is just the reverse.

An equivalence relation also defines a relation of two tuples w.r.t. a set of attributes. Following the notation in Example 6.4, we say that two tuples t_i, t_j are related w.r.t. the set of attributes X (this is: $t_i \theta_X t_j$) if $t_i(X) = t_j(X)$.

Example 6.1 We take tuples t_1, t_2 in Table 6.1, we state that $t_1 \theta_{bc} t_2$, but we do not have that $t_1 \theta_{ad} t_2$.

In this case, a set of attributes X generates an equivalence class Π_X.

Example 6.2 Again in Table 6.1, we state that $\Pi_{bc} = \{\{t_1, t_2\}, \{t_3\}, \{t_4\}\}$, and that $\Pi_a = \{\{t_1, t_3\}, \{t_2, t_4\}\}$.

6.2.2 Tolerance Relations

We introduce the concept of tolerance relation between values, which is a generalization of an equivalence relation. These two relations are extended to tuples and then used to characterize similarity and functional dependencies, respectively.

Definition 6.1 A *tolerance relation* $\theta \subseteq T \times T$ on a set T is a **reflexive** (i.e. $\forall t \in T : t \theta t$) and **symmetric** (i.e. $\forall t_i, t_j \in T : t_i \theta t_j \iff t_j \theta t_i$) relation.

A tolerance relation is somehow equivalent to the concept of **similarity** between two values, and this is how it will be used in this article.

Example 6.3 An often used tolerance relation is the *similarity* that can be defined within a set of integer values. Given two integer values v_1, v_2 and a user-defined threshold ε: $v_1 \theta v_2 \iff |v_1 - v_2| \leq \varepsilon$.

For instance, let's take a variable $Month$, defined over the months of a year. The function $\Delta_{Month}(m_1, m_2)$ defines the tolerance relation θ_{Month} such as:

$$\Delta_{Month}(m_1, m_2) = min(|m_1 - m_2|, min(m_1, m_2) + 12 - max(m_1, m_2))$$

$$m_i \theta_{Month} m_j \iff \Delta_{Month}(m_i, m_j) \leq 4$$

Then θ_{Month} is the tolerance relation that considers two values similar if they have values within 4 months of distance.

This tolerance relation on values can easily be extended to tuples.

Example 6.4 We now assume that the variable $Month$ is, in fact, an attribute of a table dataset $D = (\mathscr{U}, T)$. The function $\Delta_{Month}(t_i, t_j)$, where $t_i, t_j \in T$, defines the tolerance relation θ_{Month} such as:

$$\Delta_{Month}(t_i,t_j) = min(|t_i(Month) - t_j(Month)|, min(t_i(Month), t_j(Month)) + 12 - max(t_i(Month), t_j(Month)))$$

$$t_i \theta_{Month} t_j \iff \Delta_{Month}(t_i(Month), t_j(Month)) \leq 4$$

Then θ_{Month} is the tolerance relation that considers two **tuples** similar if they have values within 4 months of distance, this is, their value in $Month$ is similar.

A tolerance relation generates *blocks of tolerance*, among a set of tuples. A block of tolerance (w.r.t. an attribute a) is a group of tuples such that their values in attribute a are *similar*.

Definition 6.2 Given a set T, a subset $K \subseteq T$ and a tolerance relation $\theta \subseteq T \times T$. K is a *block of tolerance* of θ if:

1. $\forall t_i, t_j \in K : t_i \theta t_j$ (pairwise correspondence)
2. $\forall t_i \notin K, \exists t_j \in K : \neg(t_i \theta t_j)$ (maximality)

Given a set of tuples T and a set of attributes $\mathscr{U}$, for each attribute $x \in \mathscr{U}$, we define a tolerance relation θ_x on the values of x. The set of tolerance blocks generated by θ_x is denoted by T/θ_x.

Example 6.5 For example, when $T = \{1,2,3,4,5\}$, θ is defined as above, i.e. $v_1 \theta v_2 \iff |v_1 - v_2| \leq \varepsilon$, and $\varepsilon = 2$, then $T/\theta = \{\{1,2,3\},\{2,3,4\},\{3,4,5\}\}$.

We see that T/θ is not a partition, and this is because θ is not necessarily transitive. A partial ordering on the set of all possible tolerance relations in a set T can be defined as follows:

Definition 6.3 Let θ_1 and θ_2 two tolerance relations in the set T. We say that $\theta_1 \leq \theta_2$ if and only if $\forall K_i \in T/\theta_1 : \exists K_j \in T/\theta_2 : K_i \subseteq K_j$

This relation is a partial ordering and generates a lattice classifying tolerance relations, or, equivalently, blocks of tolerance. Given two tolerance relations, θ_1 and θ_2, the meet and the join operations in this lattice are:

Definition 6.4 Let θ_1 and θ_2 two tolerance relations in the set T.
$\theta_1 \wedge \theta_2 = \theta_1 \cap \theta_2 = \max_T(\{K_i \cap K_j \mid K_i \in T/\theta_1, K_j \in T/\theta_2\})$
$\theta_1 \vee \theta_2 = \theta_1 \cup \theta_2 = \max_T(T/\theta_1 \cup T/\theta_2)$
where $max_T(.)$ returns the set of maximal subsets w.r.t. inclusion.

Example 6.6 Based on Example 6.1, let us define the tolerance relation θ_m w.r.t. an attribute $m \in \{a,b,c,d\}$ as follows: $t_i \theta_m t_j \iff |t_i(m) - t_j(m)| \leq \varepsilon$. Then, assuming that $\varepsilon = 1$, we obtain:

$$T/\theta_a = \{\{t_1,t_3\},\{t_2,t_4\}\} \quad T/\theta_b = \{\{t_1,t_2,t_4\},\{t_3\}\}$$
$$T/\theta_c = \{\{t_1,t_2,t_3\},\{t_4\}\} \quad T/\theta_d = \{\{t_1,t_3\},\{t_2,t_4\}\}$$

We can check the following meet and join operations:

$$\theta_a \wedge \theta_b = \{\{t_1\},\{t_2,t_4\},\{t_3\}\} \quad \theta_a \vee \theta_b = \{\{t_1,t_2,t_3,t_4\}\}$$
$$\theta_a \wedge \theta_c = \{\{t_1,t_3\},\{t_2\},\{t_4\}\} \quad \theta_a \vee \theta_c = \{\{t_1,t_2,t_3,t_4\}\}$$
$$\theta_b \wedge \theta_c = \{\{t_1,t_2\},\{t_3\},\{t_4\}\} \quad \theta_b \vee \theta_c = \{\{t_1,t_2,t_3,t_4\}\}$$

We can also extend the definition of a similarity relation on a single attribute to a similarity relation to sets of attributes. Given $X \subseteq \mathscr{U}$, the similarity relation θ_X is defined as follows:

$$(t_i,t_j) \in \theta_X \iff \forall x \in X : (t_i,t_j) \in \theta_x$$

Two tuples are similar w.r.t. a set of attributes X if and only if they are similar w.r.t. *each* attribute in X.

6.3 FCA and Database Dependencies

In this section we present the two kinds of dependencies that will be discussed here: functional dependencies and similarity dependencies. In fact, the latter are a generalization of the former. As we will see, in order to switch from functional to similarity dependencies, we just need to change the implicit equivalence relation that is behind the definition of functional dependencies by a tolerance relation.

We also present the basics of FCA that are needed in order to understand some previous results concerning the characterization of FDs using FCA. We finish this section by presenting pattern structures, which is a generalization of FCA. We will use this formalism in the following section in order to present the main results in this article.

6.3.1 Functional Dependencies

Definition 6.5 ([35]) Let T be a set of tuples, and $X,Y \subseteq \mathscr{U}$. A **functional dependency (FD)** $X \rightarrow Y$ holds in T if:

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

For instance, the functional dependencies $a \to d$ and $d \to a$ hold in Table 6.1, whereas the functional dependency $a \to c$ does not hold since $t_2(a) = t_4(a)$ but $t_2(c) \neq t_4(c)$.

6.3.2 Similarity Dependencies

Similarity Dependencies are a generalization or a relaxation of Functional Dependencies. The definition of similarity dependencies is the following:

Definition 6.6 Let $X, Y \subseteq \mathscr{U}$ and let θ be a tolerance relation: $X \to Y$ is a *similarity dependency* if: $\forall t_i, t_j \in T : t_i \theta_X t_j \implies t_i \theta_Y t_j$

While a functional dependency $X \to Y$ is based on equality of values, a similarity dependency $X \to Y$ holds if each pair of tuples having *similar* values w.r.t. attributes in X has *similar values* w.r.t. attributes in Y.

This definition just replaces the equality relation (an equivalence relation) for functional dependencies with a tolerance relation, which means that transitivity is dropped. In the following sections we will see that this generalization is captured by FCA in order to characterize both kinds of dependencies.

6.3.3 Formal Concept Analysis

Formal Concept Analysis (FCA) 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 as a diagram where classes of objects/attributes and ordering relations between classes can be drawn, interpreted and used for data-mining, knowledge management and knowledge discovery [36].

We use standard definitions from [24]. 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 $(\cdot)'$:

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

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 defines the 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 $\wedge$) [24]. 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 $X \rightarrow Y$, $X,Y \subseteq M$ 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 Armstrong rules (reflexivity, augmentation, transitivity). The minimal subset of implications (in sense of its cardinality) from which all implications can be deduced with Armstrong rules is called the Duquenne-Guigues basis [25].

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.

6.3.4 Functional Dependencies as Implications

In this section we present how functional dependencies are characterized with FCA (see [3] and [24]). This is performed by transforming a table into a formal context. The implications that hold in that formal context will be equivalent to the set of functional dependencies that hold in the original table.

Again, with a table D with attributes $\mathscr{U}$ taking values in Dom, we build the formal context $\mathbb{K} = (\mathscr{B}_2(G),M,I)$, where $G = T$ and $M = \mathscr{U}$ to respect the FCA notation from [24]. $\mathscr{B}_2(G) = \{ (t_i,t_j) \mid i < j \text{ and } t_i,t_j \in T \}$ is the set of all pairs of tuples from T (excluding symmetry and reflexivity to avoid redundancy) Then, the relation I is defined as

$$(t_i,t_j)\ I\ m \Leftrightarrow t_i(m) = t_j(m), \text{ for } m \in M$$

while attributes remain the same. Figure 6.1 illustrates the transformation of the initial data to build a formal context and its concept lattice.

The number of objects of this newly built formal context is in the range of $O(|T^2|)$ (where $|T|$ is the number of tuples), so it can be significantly larger than the original set of tuples T.

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

$\mathbb{K}$	a	b	c	d
(t_1,t_2)		×	×	
(t_1,t_3)	×		×	×
(t_1,t_4)		×		
(t_2,t_3)			×	
(t_2,t_4)	×	×		×
(t_3,t_4)				

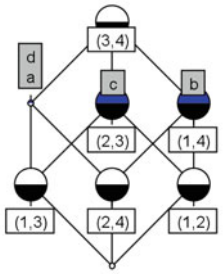

Fig. 6.1: Characterizing functional dependencies with FCA: from a set of tuples to a formal context and its concept lattice

We now explain how this concept lattice characterizes the set of all functional dependencies that hold in the table T with the following proposition:

Proposition 6.1 ([3, 24]) *A functional dependency $X \to Y$ holds in a table T if and only if $\{X\}'' = \{X,Y\}''$ in the formal context $\mathbb{K} = (\mathscr{B}_2(G), M, I)$.*

This proposition states how to test that a FD holds using the concept lattice that has been computed. For instance, let us suppose that we want to test whether a functional dependency $a \to b$ holds in the formal context of Fig. 6.1. We should test in the corresponding concept lattice if $\{a\}'' = \{a,b\}''$. In this particular case, we observe that $\{a\}'' = \{a,d\}$ and $\{a,b\}'' = \{a,b,d\}$, which means that this dependency does not hold in T. On the other hand, the dependency $ac \to d$ holds, since $\{a,c\}'' = \{a,c,d\}$ and $\{a,c,d\}'' = \{a,c,d\}$.

An interesting consequence is that the set of implications that hold in the formal context $\mathbb{K} = (\mathscr{B}_2(G), M, I)$ is syntactically equivalent to the set of functional dependencies that hold in a table T [3, 24]. By *syntactically* we mean that whenever an implication $X \to Y$ holds in $\mathbb{K}$, then the functional dependency $X \to Y$ holds in T.

6.3.5 Pattern Structures

A pattern structure is defined as a generalization of a formal context describing complex data [23]. 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 thanks to a subsumption relation $\sqsubseteq$: $\forall c,d \in D$, $c \sqsubseteq d \iff c \sqcap d = c$.

A pattern structure $(G, (D, \sqcap), \delta)$ is based on two derivation operators $(\cdot)^{\square}$:

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

These operators form a Galois connection between $(\wp(G),\subseteq)$ 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.

As for formal contexts, implications can be defined. For $c,d\in D$, the pattern implication $c\rightarrow d$ holds if $c^{\square}\subseteq d^{\square}$, i.e. the pattern d occurs in an object description if the pattern c does. Similarly, for $A,B\subseteq G$, the object implication $A\rightarrow B$ holds if $A^{\square}\sqsubseteq B^{\square}$, meaning that all patterns that occur in all objects from the set A also occur in all objects in the set B [23].

6.4 Results

In this section we show some of the most relevant results regarding the characterization of functional and similarity Dependencies with pattern structures. These results have appeared in [9, 11] for functional dependencies and in [8, 10] for Similarity Dependencies. We will see now that pattern structures are able to characterize functional and similarity dependencies in quite an elegant manner, without incurring in any extra transformation process.

6.4.1 Characterization of Functional Dependencies with Pattern Structures

The main idea behind this approach is to avoid the creation of a formal context $\mathbb{K}=(\mathscr{B}_2(G),M,I)$ (defined in Sect. 6.3.4) that, as it has already been discussed, has a size of order $O(|T^2|)$. The strategy consists in using Pattern Structures that will be built on the dataset, without any sort of transformation. Consider a dataset $D=(\mathscr{U},T)$ as a many-valued context (G,M,W,J) where $G=T$ corresponds to the set of objects ("rows"), $M=\mathscr{U}$ to the set of attributes ("columns"), $W=Dom$ the data domain ("all distinct values of the table") and $J\subseteq G\times M\times W$ a relation such that $(g,m,w)\in J$ also written $m(g)=w$ means that attribute m takes the value w for the object g [24]. In the left table of Fig. 6.2, $b(t_4)=3$.

We show how a partition pattern structure can be defined from a many-valued context (G,M,W,J) and show that its concept lattice is equivalent to the concept lattice of $\mathbb{K}=(\mathscr{B}_2(G),M,I)$ introduced above. Intuitively, formal objects of the pattern structure are the attributes of the many-valued context (G,M,W,J). Then, given an attribute $m\in M$, its description $\delta(m)$ is 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. $m(g)=m(h)$. The result is given in Fig. 6.2 (middle). As such, descriptions obey the ordering of a partition lattice as described above. It follows that (G,M,W,J) can

be represented as a pattern structure $(M,(D,\sqcap),\delta)$ where M is the set of original attributes, and $(D,\sqcap)$ is the set of partitions over G provided with the partition intersection operation $\sqcap$. An example of concept formation is given as follows, starting from set $\{a,d\} \subseteq M$:

$$\begin{aligned} \{a,d\}^{\square} &= \delta(a) \sqcap \delta(d) \\ &= \{\{t_1,t_3\},\{t_2,t_4\}\} \sqcap \{\{t_1,t_3\},\{t_2,t_4\}\} \\ &= \{\{t_1,t_3\},\{t_2,t_4\}\} \\ \{\{t_1,t_3\},\{t_2,t_4\}\}^{\square} &= \{m \in M | \{\{t_1,t_3\},\{t_2,t_4\} \sqsubseteq \delta(m)\} \\ &= \{a,d\} \end{aligned}$$

Hence, $(\{a,d\},\{\{t_1,t_3\},\{t_2,t_4\}\})$ is a pattern concept. The resulting pattern concept lattice is given in Fig. 6.2 (right).

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

m	$\delta(m) \in (D,\sqcap)$
a	$\{\{t_1,t_3\},\{t_2,t_4\}\}$
b	$\{\{t_1,t_2,t_4\},\{t_3\}\}$
c	$\{\{t_1,t_2,t_3\},\{t_4\}\}$
d	$\{\{t_1,t_3\},\{t_2,t_4\}\}$

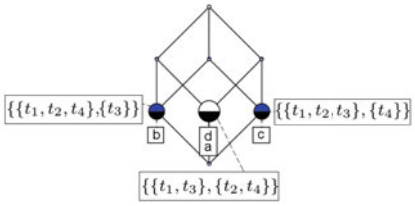

Fig. 6.2: The original data (left), the resulting pattern structure (middle) and its pattern concept lattice (right)

In the previous section, a many-valued context (G,M,W,J) was derived as the formal context $(\mathscr{B}_2(G),M,I)$ where $\mathscr{B}_2(G)$ represents any pair of objects, and $((g,h),m) \in I$ means that $m(g) = m(h)$. The resulting concept lattice is used to characterize the set of functional dependencies [24].

We also show that the lattice of pattern concepts yielded by the many-valued context (G,M,W,J) is isomorphic to the lattice of formal concepts that is yielded by the formal context $(\mathscr{B}_2(G),M,I)$:

Proposition 6.2 ([11]) *(B,A) is a pattern concept of the partition pattern structure $(M,(D,\sqcap),\delta)$ if and only if (A,B) is a formal concept of the formal context $(\mathscr{B}_2(G),M,I)$ for all $B \subseteq M, A \subseteq \mathscr{B}_2(G)$ (equivalently A is a partition on G).*

Following this equivalence, the following proposition shows that the functional dependencies that are computed in $(\mathscr{B}_2(G),M,I)$ and in (G,M,W,J) are exactly the same:

Proposition 6.3 ([11]) *A functional dependency $X \to Y$ holds in a table T if and only if: $\{X\}^{\square} = \{XY\}^{\square}$ in the partition pattern structure $(M,(D,\sqcap),\delta)$.*

Example. We consider a FD that holds in Table 6.1: $a \to d$. It is characterized from $(\mathscr{B}_2(G),M,I)$ as an attribute implication. It holds as well in $(M,\mathscr{B}_2(G),I)$ and $(M,(D,\sqcap),\delta)$ as an object implication.

6.4.2 Similarity Dependencies

In this section we present Proposition 6.4, the second main result of this article. As the reader will realize, the results in this section are very similar to those presented in the previous section. In fact, this is congruent with the fact that we are showing that the formalism of pattern structures is flexible enough to characterize not only dependencies that are based in an equivalence relation, but also with a *softer* relation, this is, a tolerance relation.

As in the previous case for functional dependencies, the dataset can be represented as a pattern structure $(\mathscr{U}, (D, \sqcap), \delta)$ where $\mathscr{U}$ is the set of original attributes. Here, $(D, \sqcap)$ is the set of sets of tolerance blocks (differently from the previous section, where we had a partition or equivalence relation) over the set of tuples T provided with the meet operation introduced in Definition 6.4. The description of an attribute $x \in \mathscr{U}$ is given by $\delta(x) = T/\theta_x$ which is given by the set of tolerance blocks.

Example 6.7 An example of concept formation in the pattern structure is given as follows. Consider the table in Example 6.1. Starting from the set $\{a, c\} \subseteq \mathscr{U}$ and assuming that $t_i \theta_x t_j \iff |t_i(x) - t_j(x)| \leq 2$ for all attributes:

$$\begin{aligned} \{a,c\}^{\square} = \delta(a) \sqcap \delta(c) &= \{\{t_1,t_3\},\{t_2,t_4\}\} \sqcap \{\{t_1,t_2,t_3\},\{t_4\}\} \\ &= \{\{t_1,t_3\},\{t_2\},\{t_4\}\} \\ \{\{t_1,t_3\},\{t_2\},\{t_4\}\}^{\square} &= \{x \in \mathscr{U} \mid \{\{t_1,t_3\},\{t_2\},\{t_4\}\} \sqsubseteq \delta(x)\} = \{a,c\} \end{aligned}$$

Hence, $(\{a,c\}, \{\{t_1,t_3\},\{t_2\},\{t_4\}\})$ is a pattern concept.

Then the pattern concept lattice allows us to characterize all similarity dependencies holding in the set of attributes $\mathscr{U}$:

Proposition 6.4 ([10]) *An similarity dependency $X \to Y$ holds in a table T if and only if: $\{X\}^{\square} = \{X,Y\}^{\square}$ in the pattern structure $(\mathscr{U}, (D, \sqcap), \delta)$.*

This result is isomorphic to Proposition 6.3. That means that the softening of the relation between tuples (equality in the case of functional dependencies, similarity in this present case) does not prevent the pattern structure from characterizing these dependencies. This is due in part to the fact that a partial order can also be defined over tolerance blocks. We do not mean that this condition is sufficient, but it is evidently necessary.

6.5 Discussion

We have revisited some results that concern the characterization of two different kinds of dependencies with pattern structures. What is the interest of using this formalism for the computation of dependencies that have already been computed using different methods not related to lattice theory? The answer is many-fold:

1. **A unified framework**. We have seen that two different kinds of dependencies may be handled using the same formalism (pattern structures). Obviously, the parameters for this formalism (objects, attributes, domain, relation) depend on the kind of dependency that we are dealing with, but as we have seen in Propositions 6.3 and 6.4, the characterization of both kinds of dependencies are equivalent.
   This is also true for other different kinds of dependencies: order-like dependencies [9, 18], acyclic join dependencies [5], degenerate multivalued dependencies [6], to name some of them. If we examine all these results, FCA offers an **encapsulation** of the semantics of each kind of dependency so that their syntax is handled by the pattern structures.
2. **Common algorithms**. As a consequence of the previous remark, we observe that potentially the same algorithms that have been used to compute formal concepts, implications, concept lattices, minimal (Duquenne-Guigues) bases can be used for all characterizations. One of the most remarkable examples is the computation of the implications that hold in a (transformed) context in Sect. 6.3.4, which are, in fact, the functional dependencies that hold in a dataset.
3. **New semantics**. Because we are using FCA and pattern structures in order to characterize dependencies, we also have structures that were unknown in database theory: formal concepts, concept lattices, etc. These objects have not been yet explored in the scenarios that we have described in this paper. Said otherwise: how can a concept lattice help a database practitioner?
4. **Different classification**. In [1, 30] there are different criteria in order to classify dependencies in the relational database model (tuple or non-tuple generating, typed or non-types, etc). However, with FCA there is no difference between the characterization of tuple-generating dependencies like multivalued dependencies [3, 7] and non tuple-generating dependencies like functional dependencies [11]. In fact, the difference is encapsulated within the relation between objects and attributes, and the difference shows up the different kinds of attributes of the formal context: partitions of the attribute set in the former case, sets of attributes in the latter. Could we devise a different classification of dependencies w.r.t. FCA?

We suggest some items for discussion that may be of interest to our FCA community as well as the database community that, eventually, may become interested in our research:

1. **Solve problems that are present in the database community**. To name just a few of them: dependencies with complex data (graphs, sequences, etc. . .), storage limitations, fast computation of queries.
2. **Do not fight the standard**. We need to admit that a standard in a long-time established community with a large number of users like that of the database community is a difficult thing to change. Even if that standard is too complex, obscure or anti-intuitive w.r.t. other formulations, the fact that this formalism has survived for such a long time in such a huge community means that it will stay there. For instance, if we take the definitions of a closure in basic texts of

database theory [1, 30], we see that the equivalent in FCA is much clearer and easy to understand (maybe, this is a biased point of view of someone who has been working with FCA for years). But even if this is the case, that community will not change their formalism by our new proposal. Therefore, instead of offering them a completely new formalism, our task should be to offer them different and *more efficient* solutions to their problems, not expecting that they would necessarily embrace FCA as a new paradigm.

3. **Have the practitioner in mind**. It is clear that some victories achieved by FCA outside our community have to do with applications rather than theory. Obviously, theory is the *sine qua non* condition for the practical solutions, but, again, what seems appealing to people outside our community is the potential use of FCA as a problem solving tool. In the case of the database community, we think that this is also true: they already have their formalisms, which are very unlikely to be changed. But they have a set of open problems that may be solved by FCA or some old problems which may be better solved using FCA.

Obviously, we just mention these points as a departure point for debate. We do not assume that all the previous points need to be an absolute truth, we just hope that they can be of interest for researchers in our community.

6.6 Conclusions

We have seen that FCA and pattern structures can characterize different kinds of dependencies in an elegant and compact way. These results are also present for different kinds of database dependencies. We have shown the advantages and some potential drawbacks that FCA offers to the database community.

We think that there has been an enormous amount of work in this fields that needs to be continued. The results presented in this paper show that there are still new lines of research that need to be explored, and that may be fruitful not only to the FCA community but, most importantly, to the database community.

Acknowledgements This research was supported by the recognition of 2017SGR-856 (MACDA) from AGAUR (Generalitat de Catalunya), and the grant TIN2017-89244-R from MINECO (Ministerio de Economía y Competitividad).

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading (MA), USA, 1995.
2. J. Baixeries. A formal concept analysis framework to model functional dependencies. In *Mathematical Methods for Learning*, 2004.

3. J. Baixeries. *Lattice Characterization of Armstrong and Symmetric Dependencies (PhD Thesis)*. Universitat Politècnica de Catalunya, 2007.
4. J. Baixeries. *A Formal Context for Symmetric Dependencies*, pages 90–105. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
5. J. Baixeries. A formal context for acyclic join dependencies. In M. Kryszkiewicz, A. Appice, D. Ślezak, H. Rybinski, A. Skowron, and Z. W. Raś, editors, *Foundations of Intelligent Systems*, pages 563–572, Cham, 2017. Springer International Publishing.
6. J. Baixeries and J. L. Balcázar. Characterization and armstrong relations for degenerate multivalued dependencies using formal concept analysis. In B. Ganter and R. Godin, editors, *ICFCA*, volume 3403 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2005.
7. J. Baixeries and J. L. Balcázar. A lattice representation of relations, multivalued dependencies and armstrong relations. In *ICCS*, pages 13–26, 2005.
8. J. Baixeries, V. Codocedo, M. Kaytoue, and A. Napoli. Characterizing approximate-matching dependencies in formal concept analysis with pattern structures. *Discrete Applied Mathematics*, 249:18 – 27, 2018. Concept Lattices and Applications: Recent Advances and New Opportunities.
9. 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.
10. J. Baixeries, M. Kaytoue, and A. Napoli. Computing similarity dependencies with pattern structures. In M. Ojeda-Aciego and J. Outrata, editors, *CLA*, volume 1062 of *CEUR Workshop Proceedings*, pages 33–44. CEUR-WS.org, 2013.
11. J. Baixeries, M. Kaytoue, and A. Napoli. Characterizing Functional Dependencies in Formal Concept Analysis with Pattern Structures. *Annals of Mathematics and Artificial Intelligence*, 72(1–2):129–149, Oct. 2014.
12. 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.
13. M. Baudinet, J. Chomicki, and P. Wolper. Constraint-generating dependencies. *J. Comput. Syst. Sci.*, 59(1):94–115, 1999.
14. 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.
15. P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.
16. L. Caruccio, V. Deufemia, and G. Polese. Relaxed functional dependencies - A survey of approaches. *IEEE Trans. Knowl. Data Eng.*, 28(1):147–165, 2016.
17. N. Caspard and B. Monjardet. The lattices of closure systems, closure operators, and implicational systems on a finite set: A survey. *Discrete Applied Mathematics*, 127(2):241–269, 2003.
18. V. Codocedo, J. Baixeries, M. Kaytoue, and A. Napoli. Characterization of Order-like Dependencies with Formal Concept Analysis. In M. Huchard and S. Kuznetsov, editors, *Proceedings of the Thirteenth International Conference on Concept Lattices and Their Applications, Moscow, Russia, July 18–22, 2016.*, volume 1624 of *CEUR Workshop Proceedings*, pages 123–134. CEUR-WS.org, 2016.
19. V. Codocedo, J. Baixeries, M. Kaytoue, and A. Napoli. Contributions to the Formalization of Order-like Dependencies using FCA. In *What can FCA do for Artificial Intelligence?*, The Hague, Netherlands, Aug. 2016.
20. J. Demetrovics, G. Hencsey, L. Libkin, and I. B. Muchnik. Normal form relation schemes: A new characterization. *Acta Cybern.*, 10(3):141–153, 1992.

21. 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.
22. A. Floratou, N. Teletia, D. J. DeWitt, J. M. Patel, and D. Zhang. Can the elephants handle the nosql onslaught? *Proc. VLDB Endow.*, 5(12):1712–1723, Aug. 2012.
23. B. Ganter and S. O. Kuznetsov. Pattern Structures and Their Projections. In *Proceedings of ICCS 2001*, Lecture Notes in Computer Science 2120, pages 129–142. Springer, 2001.
24. B. Ganter and R. Wille. *Formal Concept Analysis*. Springer, Berlin, 1999.
25. 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.
26. M. Kaytoue, S. O. Kuznetsov, A. Napoli, and S. Duplessis. Mining gene expression data with pattern structures in Formal Concept Analysis. *Information Sciences*, 181(10):1989–2001, 2011.
27. S. O. Kuznetsov. Machine learning on the basis of formal concept analysis. *Autom. Remote Control*, 62(10):1543–1564, Oct. 2001.
28. S. O. Kuznetsov. Pattern Structures for Analyzing Complex Data. In H. Sakai, M. K. Chakraborty, A. E. Hassanien, D. Slezak, and W. Zhu, editors, *RSFDGrC*, volume 5908 of *Lecture Notes in Computer Science*, pages 33–44. Springer, 2009.
29. 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.
30. D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.
31. H. Mannila and K.-J. Räihä. *The Design of Relational Databases*. Addison-Wesley, Reading (MA), USA, 1992.
32. R. Medina and L. Nourine. Conditional functional dependencies: An fca point of view. In L. Kwuida and B. Sertkaya, editors, *ICFCA*, volume 5986 of *Lecture Notes in Computer Science*, pages 161–176. Springer, 2010.
33. S. Song and L. Chen. Efficient discovery of similarity constraints for matching dependencies. *Data & Knowledge Engineering*, 87(0):146 – 166, 2013.
34. S. Song, L. Chen, and P. S. Yu. Comparable dependencies over heterogeneous data. *The VLDB Journal*, 22(2):253–274, Apr. 2013.
35. J. Ullman. *Principles of Database Systems and Knowledge-Based Systems, volumes 1–2*. Computer Science Press, Rockville (MD), USA, 1989.
36. R. Wille. Why can concept lattices support knowledge discovery in databases? *Journal of Experimental and Theoretical Artificial Intelligence*, 14(2–3):81–92, 2002.