Characterization of Order-like Dependencies with Formal Concept Analysis

Victor Codocedo, Jaume Baixeries, Mehdi Kaytoue, Amedeo Napoli

Characterization of Order-like Dependencies with Formal Concept Analysis

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

[1] Université de Lyon. CNRS, INSA-Lyon, LIRIS. UMR5205, F-69621, France.
[2] Universitat Politècnica de Catalunya. 08032, Barcelona. Catalonia.
[3] LORIA (CNRS - Inria Nancy Grand Est - Université de Lorraine), B.P. 239, F-54506, Vandœuvre-lès-Nancy.

Corresponding author: mehdi.kaytoue@insa-lyon.fr

Abstract. Functional Dependencies (FDs) play a key role in many fields of the relational database model, one of the most widely used database systems. FDs have also been applied in data analysis, data quality, knowledge discovery and the like, but in a very limited scope, because of their fixed semantics. To overcome this limitation, many generalizations have been defined to relax the crisp definition of FDs. FDs and a few of their generalizations have been characterized with Formal Concept Analysis which reveals itself to be an interesting unified framework for characterizing dependencies, that is, understanding and computing them in a formal way. In this paper, we extend this work by taking into account order-like dependencies. Such dependencies, well defined in the database field, consider an ordering on the domain of each attribute, and not simply an equality relation as with standard FDs.

Keywords: functional dependencies, order dependencies, formal concept analysis

1 Introduction

Functional dependencies (FDs) are well-known constraints in the relational model used to show a functional relation between sets of attributes [12], i.e. when the values of a set of attributes are determined by the values of another set of attributes. They are also used in different tasks within the relational data model, as for instance, to check the consistency of a database, or to guide the design of a data model [10].

Different generalizations of FDs have been defined in order to deal with imprecision, errors and uncertainty in real-world data, or simply, to mine and discover more complex patterns and constraints within data when the semantics of FDs have shown to be too restrictive for modeling certain attribute domains.

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

Table 1

For example, consider the database in Table 1 as an example[4]. Attributes of these 8 tuples are city names, month identifiers, years and average temperatures. From this table, we could expect that the value for average temperature is determined by a city name and a month of the year (e.g. the month of May in Houston is hot, whereas the month of January in Dallas is cold). Therefore, we would expect that this relationship should be somehow expressed as a (functional) dependency in the form *city name, month* $\rightarrow$ *average temperature*. However, while the average temperature is truly determined by a city and a time of the year, it is very hard that it will be exactly the same from one year to another. Instead, we can expect that the value will be *similar*, or *close* throughout different years, but rarely the same. Unfortunately, semantics of FDs is based on an equivalence relation and fail to grasp the dependencies among these attributes.

To overcome the limitations of FDs while keeping the idea that some attributes are functionally determined by other attributes, different generalizations of functional dependencies have been defined, as recently deeply reviewed in a comprehensive survey [4]. Actually, the example presented in the last paragraph is a so-called *similarity dependency* [2,4]. Several other families of dependencies exist and allow relaxing the definition of FDs on the extent part (e.g. the dependency must hold only in a subset of the tuples in a database table) or on the intent part (equality between attribute values is relaxed to a similarity or *tolerance* relation).

The definition of a variation of a functional dependency shows different problems: characterization, axiomatization and computation. Formal Concept Analysis (FCA [7]) has already been used to characterize and compute functional dependencies. Moreover, in order to overcome some of the limitations of FCA to discover FDs, a more sophisticated formalization is presented in [1] and [3] where pattern structures ([6]) were used. The same framework is used in [2] to compute similarity dependencies.

In this paper we present an FCA-based characterization of order-like dependencies, a generalization of functional dependencies in which the equality of values is replaced by the notion of order. Firstly, we show that the characterization of *order dependencies* in their general definition [8] can be achieved through a particular use of general ordinal scaling [7]. Secondly, we extend our characterization in order to support *restricted order dependencies* through which other FDs generalizations can be modeled, namely sequential dependencies and trend dependencies [4]. Finally, we present a characterization to a complex FD generalization named *lexicographical ordered dependencies* [11] showing the flexibility of our approach.

The rest of this paper is organized as follows. In Section 2 we formally introduce the definition of functional dependencies, formal concept analysis and the principle of the characterization of FDs with FCA. In Section 3, we characterize *order dependencies* in their general definition. We show that our formalization can be adapted to *restricted ordered dependencies* in Section 4 and *lexicographical ordered dependencies* [11] in Section 5 before presenting our conclusions.

[4] `http://academic.udayton.edu/kissock/http/Weather/`

2 Preliminaries

2.1 Functional dependencies

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. We define the functional notation of a tuple for a set of attributes $X \subseteq \mathcal{U}$ as follows, assuming that there exists a total ordering on $\mathcal{U}$. Given a tuple $t \in T$ and $X = \{x_1, x_2, \ldots, x_n\}$, we have: $t(X) = \langle t(x_1), t(x_2), \ldots, t(x_n)\rangle$.

Definition 1 (Functional dependency [12]). *Let T be a set of tuples (data table), and $X, Y \subseteq \mathcal{U}$. A* functional dependency (FD) $X \to Y$ *holds in T if:*

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

Example. The table on the right presents 4 tuples $T = \{t_1, t_2, t_3, t_4\}$ over attributes $\mathcal{U} = \{a, b, c, d\}$. We have that $t_2(\{a, c\}) = \langle t_2(a), t_2(c)\rangle = \langle 4, 4\rangle$. Note that the set notation is usually omitted and we write ab instead of $\{a, b\}$. In this example, the functional dependency $d \to c$ holds and $a \to c$ does not hold.

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	8

Table 2

2.2 Formal Concept Analysis (FCA)

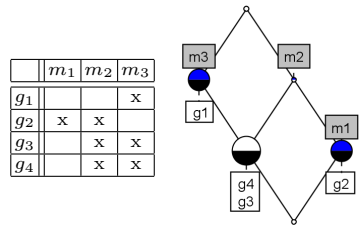
Let G and M be arbitrary sets, respectively called *objects* and *attributes*, and $I \subseteq G \times M$ an arbitrary binary relation: $(g, m) \in I$ is interpreted as "g has attribute m". (G, M, I) is called a formal context. The two following derivation operators $(\cdot)'$ define a Galois connection between the powersets of G and M.

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

For $A \subseteq G$, $B \subseteq M$, a pair (A, B) such that $A' = B$ and $B' = A$, is called a *(formal) concept* while A is called the *extent* and the set B the *intent* of the concept. Concepts are partially ordered by $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2$ ($\Leftrightarrow B_2 \subseteq B_1$): the set of all formal concepts forms a complete lattice called the *concept lattice* of the formal context (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 Armstrong rules (reflexivity, augmentation, transitivity).

Example. The table on the left presents a formal context: we have $(\{g_3\}'', \{g_3\}') = (\{g_3, g_4\}, \{m_2, m_3\})$ and the implication $m_1 \to m_2$. Its concept lattice representation involves *reduced labeling*: each node is a concept, lines represent partial ordering while an attribute (resp. object) label is inherited from the top (resp. the bottom).

	m_1	m_2	m_3
g_1			x
g_2	x	x	
g_3		x	x
g_4		x	x



2.3 Characterization of Functional Dependencies with FCA

It has been shown in previous work that functional dependencies, can be characterized with FCA. For example, Ganter & Wille [7] presented a data transformation of the initial set of tuples into a formal context. In this context, implications are in 1-to-1 correspondence with the functional dependencies of the initial dataset. In Figure 1, we illustrate this characterization with the set of tuples of Table 2. Each possible pair of tuples gives rise to an object in the formal context. Attributes remain the same. An object, say (t_i, t_j), has an attribute m iff $t_i(m) = t_j(m)$. The concept lattice is given on the right hand side of this figure: there are two implications, namely $d \to c$ and $d \to a$, which are also the functional dependencies in the original set of tuples.

However, this approach implies that a formal context much larger than the original dataset must be processed. It was then shown that this formal context can actually be encoded with a pattern structure [6]: each attribute of the original dataset becomes an object of the pattern structure and is described by a partition on the tuple set. Actually, each block of the partition is composed of tuples taking the same value for the given attribute [9]. For example, in Table 2, the partition describing a is $\{\{t_1, t_3\}, \{t_2, t_4\}\}$. Then, the implications in the pattern concept lattice are here again in 1-to-1 correspondence with the functional dependencies of the initial dataset [3]. What is important to notice is that this formalization is possible as a partition is an *equivalence relation*: a symmetric, reflexive and transitive binary relation.

In [2], another kind of dependencies was formalized in a similar way, i.e. similarity dependencies, where the equality relation is relaxed to a similarity relation when comparing two tuples. An attribute is not anymore described by a partition, but by a tolerance relation, i.e. a symmetric, reflexive, but not necessarily transitive binary relation (*"the friends of my friends are not necessarily my friends"*). Each original attribute is then described by a set of tolerance blocks, each being a maximal set of tuples that have pairwise similar values (instead of equal values for classical dependencies).

This way of characterizing FDs and similarity dependencies actually fails for order dependencies, as the relation in this case is not symmetric: it is neither an equality nor a similarity but a partial order in the general case.

3 Characterization of Order Dependencies with FCA

Although functional dependencies are used in several domains they cannot be used to express some relationships that exist in data. Many generalizations have been proposed and we focus in this article on order dependencies [8,4]. Such dependencies are based on the *attribute-wise* order on tuples. This order assumes that each attribute follows a partial order associated to the values of its domain. For the sake of generality, we represent this order with the symbol $\sqsubseteq_x$ for all $x \in \mathcal{U}$. In practice, this symbol will be instantiated by intersections of any partial order on the domain of this attribute, as, for instance, $<, \leq, >, \geq$, etc.

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	8

$\mathbb{K}$	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)				

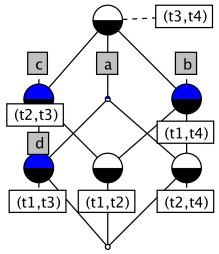


Fig. 1: Characterizing functional dependencies with FCA.

We remark that this order on the set of values of a *single* attribute does not need to be a total order, although in many different instances, like numeric or character strings domains, this will be the case. In the following, we formalize operator $\sqsubseteq_x$ (Definition 2) and define accordingly order dependencies (Definition 3).

Definition 2 (Attribute-wise ordering). *Given two tuples $t_i, t_j \in T$ and a set of attributes $X \subseteq \mathcal{U}$, the attribute-wise order of these two tuples on X is:*

$$t_i \sqsubseteq_X t_j \Leftrightarrow \forall x \in X : t_i[x] \sqsubseteq_x t_j[x]$$

This definition states that one tuple is *greater* –in a sense involving the order of all attributes– than another tuple if their attribute-wise values meet this order. This operator induces a partial order $\Pi_X = (T, \prec_X)$ on the set T of tuples.

Definition 3 (Order dependency). *Let $X, Y \subseteq \mathcal{U}$ be two subsets of attributes in a dataset T. An order dependency $X \to Y$ holds in T if and only if:*

$$\forall t_i, t_j \in T : t_i \sqsubseteq_X t_j \to t_i \sqsubseteq_Y t_j$$

Example. Consider the table on the right with six tuples and three attributes. Taking $\sqsubseteq_a$, $\sqsubseteq_b$ and $\sqsubseteq_c$ defined as the ordering $\leq$. The orders induced by the sets of attributes $\{a\}$,$\{b\}$,$\{c\}$ and $\{a, b\}$ are:

$\Pi_a = (T, \prec_a) = \{\{t_1\} \prec \{t_2\} \prec \{t_3, t_6\} \prec \{t_5\} \prec \{t_4\}\}$
$\Pi_b = (T, \prec_b) = \{\{t_5\} \prec \{t_1, t_4\} \prec \{t_3\} \prec \{t_2\} \prec \{t_6\}\}$
$\Pi_{ab} = (T, \prec_{ab}) = \{\{t_1\} \prec \{t_2\} \prec \{t_6\};$
$\{t_1\} \prec \{t_3\};$
$\{t_1\} \prec \{t_4\};$
$\{t_5\} \prec \{t_4\}\}$
$\Pi_c = (T, \prec_c) = \{\{t_1\} \prec \{t_2\} \prec \{t_3, t_6\} \prec \{t_5\} \prec \{t_4\}\}$

id	a	b	c
t_1	1	3	1
t_2	2	7	2
t_3	3	4	4
t_4	5	3	9
t_5	4	2	5
t_6	3	8	4

Table 3

These orders are such that the order dependency $\{a, b\} \to \{c\}$ holds. Remark that Definition 3 is generic since the orders that are assumed for each attribute

need to be instantiated: we chose $\leq$ in this example for all attributes, while taking the equality would produce standard *functional dependencies*.

To achieve the characterization of order dependencies with FCA, we propose to represent the partial order $\Pi_X = (T, \prec_X)$ associated to each subset of attribute $X \subseteq \mathcal{U}$ as a formal context $\mathbb{K}_X$ (a binary relation on $T \times T$ thanks to a general ordinal scaling [7]). Then, we show that an order dependency $X \to Y$ holds iff $\mathbb{K}_X = \mathbb{K}_{XY}$.

Definition 4 (General ordinal scaling of the tuple set). *Given a subset of attributes $X \subseteq \mathcal{U}$ and a table dataset T, we define a formal context for $\Pi_X = (T, \prec_X)$ (the partial order it induces) as follows:*

$$\mathbb{K}_X = (T, T, \sqsubset_X)$$

*where $\sqsubset_X = \{(t_i, t_j) \mid t_i, t_j \in T, t_i \sqsubseteq_X t_j\}$. This formal context is the **general ordinal scale** of Π_X [7]. All formal concepts $(A, B) \in \mathbb{K}_X$ are such that A is the set of lower bounds of B and B is the set of upper bounds of A. Its concept lattice is the smallest complete lattice in which the order Π_X can be order embedded.*

This way to characterize a partial order is only one among several possibilities. However, the choice of formal contexts is due to their versatility, since they can characterize binary relations, hierarchies, dependencies, different orders [7] and graphs [5]. In the next section we will see how this versatility allows us to generalize similarity dependencies. Given the set of attributes $X \subseteq \mathcal{U}$, an associated partial order $\Pi_X = (T, \prec_X)$ and the formal context $(T, T, \sqsubset_X)$, it is easy to show that the later is a composition of contexts defined as: $(T, T, \sqsubset_X) = (T, T, \bigcap_{x \in X} \sqsubset_x)$.

We can now propose a characterization of order dependencies with FCA.

Proposition 1. *An order dependency $X \to Y$ holds in T iff $\mathbb{K}_X = \mathbb{K}_{XY}$.*

Proof. Recall that $\mathbb{K}_{XY} = (T, T, \sqsubset_{XY}) = (T, T, \sqsubset_X \cap \sqsubset_Y)$. We have that

$$\begin{aligned} X \to Y &\iff \sqsubset_X = \sqsubset_X \cap \sqsubset_Y \\ &\iff \sqsubset_X \subseteq \sqsubset_Y \\ &\iff \forall t_i, t_j \in T,\ t_i \sqsubseteq_X t_j \to t_i \sqsubseteq_Y t_j \end{aligned}$$

Example. To calculate $(T, T, \sqsubset_{ab})$, we just need to calculate $(T, T, \sqsubset_a \cap \sqsubset_b)$, as illustrated in the example below.

$\sqsubset_a$	t_1	t_2	t_3	t_4	t_5	t_6
t_1		×	×	×	×	×
t_2			×	×	×	×
t_3				×	×	
t_4						
t_5				×		
t_6				×	×	

Table 4: $(T, T, \sqsubset_a)$

$\sqsubset_b$	t_1	t_2	t_3	t_4	t_5	t_6
t_1		×	×			×
t_2						×
t_3		×				×
t_4		×	×			×
t_5	×	×	×	×		×
t_6						

Table 5: $(T, T, \sqsubset_b)$

$\sqsubset_{ab}$	t_1	t_2	t_3	t_4	t_5	t_6
t_1		×	×			×
t_2						×
t_3						
t_4						
t_5				×		
t_6						

Table 6: $(T, T, \sqsubset_{ab})$

The fact that the order dependency $\{a, b\} \to \{c\}$ holds can be illustrated with the formal contexts in Tables 7,8 and 9. We have indeed that $\mathbb{K}_{ab} = \mathbb{K}_{abc}$.

$\sqsubset_c$	t_1	t_2	t_3	t_4	t_5	t_6
t_1		×	×	×	×	×
t_2			×	×	×	×
t_3				×	×	×
t_4						
t_5				×		
t_6				×	×	

Table 7: $(T, T, \sqsubset_c)$

$\sqsubset_{ab}$	t_1	t_2	t_3	t_4	t_5	t_6
t_1		×	×			×
t_2						×
t_3						
t_4						
t_5				×		
t_6						

Table 8: $(T, T, \sqsubset_{ab})$

$\sqsubset_{abc}$	t_1	t_2	t_3	t_4	t_5	t_6
t_1		×	×			×
t_2						×
t_3						
t_4						
t_5				×		
t_6						

Table 9: $(T, T, \sqsubset_{abc})$

Order dependencies and other FDs generalizations. We have seen that the definition of order dependencies replaces the equality condition present in FDs or other similarity measures present in other dependencies, by an order relation. This may suggest that order dependencies and other kinds of FDs generalizations are structurally very similar, whereas this is not the case. Functional dependencies generate a reflexive, symmetric and transitive relation in the set of tuples, i.e. an equivalence relation. Then the set of tuples can be partitioned into *equivalence classes* that are used to characterize and compute the set of FDs holding in a dataset, as presented in a previous work [3].

In the generalization of functional dependencies that replaces the equality condition by a *similarity* measure or a distance function, this measure generates a symmetric relation in the set of tuples, but not necessarily a transitive relation. In turn, this implies that the set of tuples can be partitioned into *blocks of tolerance* instead of equivalence classes, as shown in [2].

In this article, the novelty is that we are dealing with a transitive relation, but not necessarily a symmetric relation. That means that we are not dealing with equivalence classes nor blocks of tolerance any longer, but, precisely, with orders. And since the characterization of these dependencies cannot be performed in terms of equivalence classes nor blocks of tolerance, will use a more general approach: *general ordinal scaling*.

4 Characterization of Restricted Order Dependencies

	Time	*People_waiting*
t_1	10:00	101
t_2	10:20	103
t_3	10:40	105
t_4	11:00	77
t_5	11:20	80
t_6	11:40	85

Table 10

Order dependencies allow taking into account the ordering of the values of each attribute when looking for dependencies in data. However, violations of the ordering due to value variations should sometimes not be considered in many real world scenarios. Consider the example given in Table 10: it gives variations on the number of people waiting at a bus station over time. In such a scenario we can expect that more people will be waiting in the station as time moves on ($People\_waiting \to Time$). However, at some point, a bus arrives and the number of people waiting decreases and starts increasing again. It is easy to observe that the order dependency $People\_waiting \to Time$

does not hold as we have the counter-example:

$$t_4 \sqsubseteq_{People\_waiting} t_3 \text{ and } t_3 \sqsubseteq_{Time} t_4$$

However, the gap between the values 77 and 105 is significant enough to be considered as a different instance of the ordering. We can formalize this idea by introducing a *similarity threshold* $\theta = 10$ for the attribute *People_waiting* such that the ordering between values is checked iff the difference is smaller than θ. In this way, the previous counter-example is avoided (*restricting* the binary relation) along with any other counter-example and we have that the restricted order dependency $People\_waiting \rightarrow Time$ holds.

We now formalize the tuple ordering relation, and consequently the notion of restricted order dependencies.

Definition 5. *Given two tuples $t_i, t_j \in T$ and a set of attributes $X \subseteq \mathcal{U}$, the attribute-wise order of these two tuples on X is:*

$$t_i \sqsubseteq^*_x t_j \Leftrightarrow \forall x \in X : 0 \leq t_j[x] - t_i[x] \leq \theta_x$$

Definition 6. *Let $X, Y \subseteq \mathcal{U}$ two sets of attributes in a table T such that $|T| = n$, and let θ_X, θ_Y be thresholds values of tuples in X and Y respectively. A restricted order dependency $X \rightarrow Y$ holds in T if and only if:*

$$t[X] \sqsubseteq^*_X t'[X] \rightarrow t[Y] \sqsubseteq^*_Y t'[Y]$$

Using these definitions we can encode the tuple ordering relations as formal contexts for any subset of attributes $X \subseteq \mathcal{U}$. Indeed, the binary relations between tuples by operator $\sqsubseteq^*_X$ can be encoded in a formal context $\mathbb{K}^*_X = (T, T, \sqsubseteq^*_X)$ which in turn, can be composed from single attributes $x \in \mathcal{U}$ as follows:

$$\sqsubseteq^*_X = \bigcap_{x \in X} \sqsubseteq^*_x$$

Moreover, we can use the same rationale we used to mine order dependencies to find restricted order dependencies.

Proposition 2. *A restricted order dependency $X \rightarrow Y$ holds in T iff*

$$X \rightarrow Y \iff \mathbb{K}^*_X = \mathbb{K}^*_{XY}$$

Proof. This proposition can be proved similarly to Proposition 1.

Example. For the previous example, we calculate the corresponding formal contexts shown in Tables 11 and 12 ($\sqsubseteq^*_{Tm}$ for *Time*, and $\sqsubseteq^*_{Pp}$ for *People_waiting*). It is easy to observe that the restricted order dependency $People\_waiting \rightarrow Time$ holds as we have that $\mathbb{K}^*_{Pp} = \mathbb{K}^*_{Pp,Tm}$.

$\sqsubseteq^*_{Tm}$	t_1	t_2	t_3	t_4	t_5	t_6
t_1	×	×	×	×	×	×
t_2		×	×	×	×	×
t_3			×	×	×	×
t_4				×	×	×
t_5					×	×
t_6						×

Table 11: $(T, T, \sqsubseteq^*_{Tm})$

$\sqsubseteq^*_{Pp}$	t_1	t_2	t_3	t_4	t_5	t_6
t_1	×	×	×			
t_2		×	×			
t_3			×			
t_4				×	×	×
t_5					×	×
t_6						×

Table 12: $(T, T, \sqsubseteq^*_{Pp})$

$\sqsubseteq^*_{Tm,Pp}$	t_1	t_2	t_3	t_4	t_5	t_6
t_1	×	×	×			
t_2		×	×			
t_3			×			
t_4				×	×	×
t_5					×	×
t_6						×

Table 13: $(T, T, \sqsubseteq^*_{Tm,Pp})$

Restricted order dependencies and other FDs generalizations. Similarity dependencies (SDs) generalize functional dependencies through the use of a tolerance relation instead of an equivalence relation between values of tuples for a given set of attributes. A tolerance relation is a reflexive, symmetric and non-transitive binary association between two tuples given a threshold θ. In a nutshell, a SD is established between two tuples if their values are within a given *distance* controlled by the threshold. Such dependencies were studied in a previous work [2]. However, from the perspective of order dependencies, we can request that such distance has a certain polarity. As we have previously discussed, order dependencies arise from anti-symmetric, not necessarily reflexive, and transitive binary relations $(<, \leq)$. Then, it can be expected that using a *threshold of distance* θ between tuple values for a given set of attributes requires an antisymmetric, non-transitive relation between the values of tuples w.r.t. a set of attributes X, that we have defined as $\sqsubseteq^*_X$.

Observe that the difference between Definition 5 and tolerance relations is the drop of the absolute value for $t_j[x] - t_i[x]$ and the requirement that this is a positive number, i.e. $t_i[x] < t_j[x], \forall x \in X$. There is an important difference between this setting and SDs. While in SDs the threshold θ is used to *relax* the strict equivalence condition of standard functional dependencies, from the perspective of order dependencies the threshold is actually used to *restrict* tuple relations.

Restricted order dependencies have the potential to implement some other generalizations of FDs such as sequential dependencies and trend dependencies [4]. The latter is actually a particular case of restricted order dependencies where the threshold is applied to an attribute not contained in the attributes of the dependency. Instead, it is applied to a *time* attribute that allows defining a *snapshot* of the database. In sequential dependencies, the antecedent is a mapping (ρ) of a set of attributes with a *complete unrestricted* order (without a threshold). Currently, we are able to support some instances of sequential dependencies when the mapping is symmetric ($\rho(XY) = \rho(YX)$). Details on this matter has been left out from this paper for space reasons. Rather, in the following section we describe another dependency which is not symmetric, namely lexicographical ordered dependencies (LODs), that exemplifies the flexibility of our approach to support complex dependency definitions.

5 Lexicographical ordered dependencies LODs

LODs use the notion of lexicographical ordering in a rather unconventional manner[5]. While it could be expected that they compare the values of different attributes using lexicographical order, instead new descriptions (or *projections*) are composed from the Cartesian product of attribute domains on which the lexicographical order is applied [11]. Consequently, the order in which we compose new descriptions becomes relevant.

[5] Consider lexicographical order as the order of words in a dictionary.

For example, in Table 14 we compose a description using the ordered set $X = \langle b, e\rangle$ such that the new descriptions or *projections* of tuples t_1, t_2 on X are $t_1^X =< 3, 5 >$ and $t_2^X =< 4, 0 >$ respectively. It is clear that t_1^X is lexicographically lower than t_2^X. However, considering $Y = \langle e, b\rangle$ which is the inverse of X, we have $t_1^Y = 53$ and $t_2^Y = 04$ where t_2^Y is lexicographically lower than t_1^Y. Definition 7 formalizes lexicographical ordering for tuple projections.

Definition 7 ([11]). *Let $X \subseteq M$ be an ordered set, such that $n = |X|$, and let $[1, n]$ be the set of indices of the ordered set X. A lexicographical ordering on X, denoted by $\leq_X^l$ is defined for t_1^X, t_2^X as $t_1^X \leq_X^l t_2^X$, if either:*

1. *$\exists k \in [1, n]$ s.t. $t_1^X[k] < t_2^X[k]$ and $t_1^X[j] = t_2^X[j]$ with $j \in [1, k[$.*
2. *$t_1^X[i] = t_2^X[i]$, $\forall i \in [1, n]$*

The main difference between LODs and standard order dependencies is that LODs are established over *ordered sets* and thus, the LOD $\langle a, b\rangle \rightsquigarrow \langle c\rangle$ does not imply that $\langle b, a\rangle \rightsquigarrow \langle c\rangle$ holds, where $\rightsquigarrow$ is used to denote a LOD. Definition 8 formalizes lexicographical order dependencies.

Definition 8 ([11]). *Let $X, Y \subseteq \mathcal{U}$ be two ordered attribute sets. A LOD, $X \rightsquigarrow Y$ is satisfied iff for all $t_1, t_2 \in r$, $t_1^X \leq_X^l t_2^X$ implies that $t_1^Y \leq_Y^l t_2^Y$.*

In Table 14, we have the LOD $\langle c, a, b\rangle \rightsquigarrow \langle d, e\rangle$ which can be verified as follows. Let $X = \langle c, a, b\rangle$ and $Y = \langle d, e\rangle$.

$$(t_1^X = 123) \leq_X^l (t_2^X = 124) \rightarrow (t_1^Y = 45) \leq_X^l (t_2^Y = 60)$$

	a	b	c	d	e
t_1	2	3	1	4	5
t_2	2	4	1	6	0

Table 14: Example

$\leq_{\langle e,d\rangle}^l$	t_1	t_2
t_1	×	
t_2	×	×

Table 15: $\mathbb{K}_{ed}^l$

$\leq_{\langle d,e\rangle}^l$	t_1	t_2
t_1	×	×
t_2		×

Table 16: $\mathbb{K}_{de}^l$

As pointed out in [11], a LOD between single attributes is necessarily an order dependency (with a single attribute there is only one order). Furthermore, given point 2 of Definition 7, all functional dependencies are also LODs. This includes the permutations of the antecedent and the consequent of a FD.

In our setting, we have described that a context can be build to encode tuple relations for a given order operator (e.g. $\sqsubseteq_X, \sqsubseteq_X^*$) w.r.t. a set of attributes X. Regarding LODs, this cannot be the case as the set of attributes X is required to be *ordered*, meaning that the context $\mathbb{K}_{xy}^l$ is not necessarily the same as the context $\mathbb{K}_{yx}^l$ for $x, y \in \mathcal{U}$ ($\mathbb{K}^l$ indicates a formal context encoding a lexicographical ordering $\leq^l$). For example, from Table 14 we can build a formal context encoding $\leq_{\langle e,d\rangle}^l$ (shown in Table 15) where $t_2 \leq_{\langle e,d\rangle}^l t_1$, and a different formal context encoding $\leq_{\langle d,e\rangle}^l$ (shown in Table 16) where $t_1 \leq_{\langle d,e\rangle}^l t_2$.

Nevertheless, a close inspection to a generic LOD $X \rightsquigarrow Y$ reveals that it requires a series of order-like dependencies to hold to be satisfied. For example, if $X \rightsquigarrow Y$ then the functional dependency $X \rightarrow Y$ holds. We can prove this

as follows. Consider two tuples such that $t_i^X = t_j^X$ (their projections w.r.t. X are equivalent). Then, $t_i \leq_X^l t_j$ and $t_j \leq_X^l t_i$ which, if $X \rightsquigarrow Y$ holds, implies that $t_i \leq_Y^l t_j$ and $t_j \leq_Y^l t_i$ which is the same as $t_i^Y = t_j^Y$. Clearly, $t_i^X = t_j^X$ regardless the order of attributes in X and thus we have a functional dependency $X \to Y$. Now, consider the first attribute of X, x_1 and the first attribute of Y, y_1. Necessarily, $t[x_1] < t'[x_1] \to t[y_1] \leq t'[y_1]$ which is an order-like dependency between x_1 and y_1. Similar analysis can be used to obtain sufficient rules so $X \rightsquigarrow Y$.

For the sake of brevity, we describe a simple algorithm verifying that the LOD $X \rightsquigarrow Y$ holds by checking a *cascade* of order-like dependencies obtained using the previous analysis. Table 5 presents an example of this algorithm applied to a table containing a LOD. For each attribute $x \in \mathcal{U}$ we generate three different formal contexts, namely $\mathbb{K}_x^= = (T, T, =_x)$, $\mathbb{K}_x^< = (T, T, <_x)$ and $\mathbb{K}_x^\leq = (T, T, \leq_x)$. Then, we proceed as follows:

- Check functional dependency $X \to Y$
- For the i-th element of Y, y_i:
  - Build $\mathbb{K}_\psi = \mathbb{K}_{y_1}^= \cap \mathbb{K}_{y_2}^= \cap \mathbb{K}_{y_3}^= \cap \cdots \cap \mathbb{K}_{y_{i-1}}^=$
  - For the j-th element of X, x_j:
    - Build $\mathbb{K}_\chi = \mathbb{K}_{x_1}^= \cap \mathbb{K}_{x_2}^= \cap \mathbb{K}_{x_3}^= \cap \cdots \cap \mathbb{K}_{x_{j-1}}^=$
    - Check the order-like dependency $(\mathbb{K}_\chi \cap \mathbb{K}_{x_j}^< \cap \mathbb{K}_\psi) \subseteq \mathbb{K}_{y_i}^\leq$

Checking $\langle a, b\rangle \rightsquigarrow \langle c, d\rangle$:

- $\mathbb{K}_{ab}^=$ is empty and the FD $ab \to cd$ holds.
- For the first element of $\langle c, d\rangle$:
  - $\mathbb{K}_a^< = \{(t_1, t_2)\} \subseteq \mathbb{K}_c^\leq = \{(t_1, t_2), (t_1, t_3), (t_2, t_3), (t_3, t_2)\}$
  - $\mathbb{K}_a^= \cap \mathbb{K}_b^< = \{(t_2, t_3)\} \subseteq \mathbb{K}_c^\leq = \{(t_1, t_2), (t_1, t_3), (t_2, t_3), (t_3, t_2)\}$
- For the second element of element of $\langle c, d\rangle$:
  - $\mathbb{K}_a^< \cap \mathbb{K}_c^= = \emptyset \subseteq \mathbb{K}_d^\leq = \{(t_2, t_3)\}$
  - $\mathbb{K}_a^= \cap \mathbb{K}_b^< \cap \mathbb{K}_c^= = \{(t_2, t_3)\} \subseteq \mathbb{K}_d^\leq = \{(t_2, t_3)\}$

	a	b	c	d
t_1	1	?	1	?
t_2	2	1	2	1
t_3	2	2	2	2

Table 17: Example

6 Conclusion

Different generalizations of functional dependencies have been defined. The definition of a new kind of generalization of functional dependencies needs to cover two different aspects: axiomatization and computation.

We have presented a characterization of order dependencies with FCA, which can be potentially extended to other types of order-like dependencies, and are used in many fields in database theory, knowledge discovery and data quality.

These dependencies are part of a set of functional dependencies generalizations where equality condition is replaced with a more general relation. In some cases, the equality is replaced by an approximate measure, in other cases, like in order dependencies, by an order relation.

We have seen that order dependencies are based on a transitive, but not necessarily symmetric relation, contrasting approximate dependencies, which are

based on a symmetric, but not necessarily transitive relation. It is precisely this formalization in terms of FCA that allows us to find these structural differences between these types of dependencies. We have also seen that this same characterization can be extended to other kinds of approximate dependencies.

This characterization allows us to cover also the computation of these dependencies: they can use the different algorithms that already exist in FCA.

This present work needs to be extended to other kinds of order-like dependencies, and some experimentation needs to be performed in order to verify the computational feasibility of this approach.

Acknowledgments. This research work has been supported by the SGR2014-890 (MACDA) project of the Generalitat de Catalunya, and MINECO project APCOM (TIN2014-57226-P) and partially funded by the French National Project FUI AAP 14 Tracaverre 2012-2016.

References

1. 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.
2. 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.
3. 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.
4. L. Caruccio, V. Deufemia, and G. Polese. Relaxed functional dependencies - A survey of approaches. *IEEE Trans. Knowl. Data Eng.*, 28(1):147–165, 2016.
5. S. Ferré. A Proposal for Extending Formal Concept Analysis to Knowledge Graphs. In *Formal Concept Analysis*, volume LNCS 9113, pages 271–286, Nerja, Spain, June 2015.
6. 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.
7. B. Ganter and R. Wille. *Formal Concept Analysis*. Springer, Berlin, 1999.
8. S. Ginsburg and R. Hull. Order dependency in the relational model. *Theoretical Computer Science*, 26(1):149 – 195, 1983.
9. 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.
10. H. Mannila and K.-J. Räihä. *The Design of Relational Databases*. Addison-Wesley, Reading (MA), USA, 1992.
11. W. Ng. Ordered functional dependencies in relational databases. *Information Systems*, 24(7):535 – 554, 1999.
12. J. Ullman. *Principles of Database Systems and Knowledge-Based Systems, volumes 1–2*. Computer Science Press, Rockville (MD), USA, 1989.