Three Views on Dependency Covers from an FCA Perspective

Jaume Baixeries, Victor Codocedo, Mehdi Kaytoue, Amedeo Napoli

Three Views on Dependency Covers from an FCA Perspective

Jaume Baixeries[1][0000−0003−2896−2247], Victor Codocedo[2][0000−0002−6360−2039], Mehdi Kaytoue[3][0000−0002−1569−5242], and Amedeo Napoli[4][0000−0001−5236−9561]

[1] Universitat Politècnica de Catalunya, Barcelona, Catalonia.
[2] Instituto para la Resiliencia ante Desastres, Chile
[3] Université de Lyon. CNRS, INSA-Lyon, LIRIS, Lyon, France
[4] Université de Lorraine, CNRS, LORIA, Nancy, France

Abstract. Implications in Formal Concept Analysis (FCA), Horn clauses in Logic, and Functional Dependencies (FDs) in the Relational Database Model, are very important dependency types in their respective fields. Moreover, they have been proved to be equivalent from a syntactical point of view. Then notions and algorithms related to one dependency type in a field can be reused and applied to another dependency type in the other field. One of these notions is that of *cover*, also known as a *basis*, i.e., a compact representation of a complete set of implications, FDs, or Horn clauses. Although the notion of cover exists in the three fields, the characterization and the related uses of a cover are different. In this paper, we study and compare, from an FCA perspective, the principles on which rely the most important covers in each field. Finally, we discuss some open questions that are of interest in the three fields, and especially to the FCA community.

Keywords: Functional dependencies · Implications · Horn Clauses · Dependency Covers · Closure.

1 Introduction and Motivation

A **dependency** is a relation between sets of attributes in a dataset. In this paper, they are represented as $X \to Y$, where the type of the subsets of attributes X and Y, and the semantics of $\to$ may vary w.r.t. the context. There are many different kinds of dependencies: complete and comprehensive surveys, from a Relational Database Theory perspective, can be found in [18] and in [13]. Here, we focus on those dependencies that follow the so called Armstrong axioms, this is, reflexivity, augmentation and transitivity, which appear in different fields of computer science: *functional dependencies* (FDs) in the Relational Database Model, *Horn clauses* in logic and logic programming, and *implications* in Formal Concept Analysis.

Functional dependencies [27] are of paramount importance in the Relational Database Model (RDBM), where they are used to express constraints or rules that need to hold in a database, to help the design of a database or to

check for errors and inconsistencies. A set of **Horn clauses** [16] is a special case of Boolean functions that are crucial in logic programming [19,23] and artificial intelligence (see [9] for a detailed explanation). **Implications** are at the core of Formal Concept Analysis (FCA) where they are used to model and deduce relevant information that is contained in a formal context [15,14].

Although all three of them appear in different fields, have been applied on different kinds of data and have been used for different purposes, they all share the same axioms, which means that, from a syntactical point of view, they are all equivalent. More specifically, the equivalence between functional dependencies and Horn clauses is presented in [25,17] (see also [9] for a more detailed explanation). The equivalence between functional dependencies and implications is explained in [15] and the equivalence between implications and Horn clauses is explained in Section 5.1 in [14] as well as in [8]. These equivalences allow us to talk in a generic way of **Armstrong dependencies** or, simply, **dependencies**.

One of the consequences of this equivalence is the transversality of concepts, problems and algorithms between these three fields. One of the most typical examples is the decision problem of the **logical implication** which consists in, given a set of dependencies Σ and a single dependency σ, to determine whether σ is logically entailed by Σ, that is, $\Sigma \models \sigma$. Entailment means that σ can be derived from Σ by the iterative application of the Armstrong axioms. This problem is named **implication problem** in the RDBM [20,1] and FCA fields, and **deduction** in logic [9]. It is of capital interest in all three fields. In the RDBM it allows to test whether two different sets of functional dependencies are equivalent [13], and it also allows to compute a more succinct set of functional dependencies, which is relevant to assist the design process of databases [4,22]. In logic the deduction problem is used to check whether a logical expression is consistent w.r.t. a knowledge base and to compute the prime implicants of a Horn function [9]. In Formal Concept Analysis this problem is used, for instance, in attribute exploration [14], which consists in creating a data table (context) in agreement w.r.t. a set of attributes and a set of objects, and also for computing the Duquenne-Guigues basis [11].

Roughly speaking, the computation of the logical implication problem $\Sigma \models X \rightarrow Y$ is performed by iterating over Σ and applying the Armstrong axioms to infer new dependencies until a positive or negative answer is found. However, this problem can be reduced to the computation of the **closure** of X with respect to Σ ($\text{closure}_\Sigma(X)$). This closure returns the largest set such that $\Sigma \models X \rightarrow \text{closure}_\Sigma(X)$ holds. Therefore, the implication problem $\Sigma \models X \rightarrow Y$ boils down to testing whether $Y \subseteq \text{closure}_\Sigma(X)$.

As an example of transversality, the algorithm that computes $\text{closure}_\Sigma(X)$ appears in most of the main database textbooks, where it is called **closure** [1,20,27], and also in logic, where it is called **forward chaining** [9], while in FCA the same algorithm that first appeared in the RDBM is discussed and reused in [14]. An improved version of Closure is Linclosure [4]. Both Closure and Linclosure have two parameters: a set of attributes X and a set of dependencies Σ. The performance of both algorithms is first determined by their worst-case complexity:

quadratic in the case of Closure, linear in the case of Linclosure, always with respect to the size of Σ, although the nature and the size of Σ greatly affect the performance of both algorithms outside of the worst case.

Another "transversal notion" which is present in all three fields is the notion of **cover**. In general terms, it is not suitable to handle the set Σ of all dependencies that hold, because of its potential large size, but rather a subset of Σ that contains the same information and that is significantly smaller in size. By *"containing the same information"* we mean that this subset may generate, thanks to the application of the Armstrong axioms, the complete set Σ. This compact and representative subset is called "cover" in the RDBM, "basis" in FCA and "set of prime implicants" in logic. Moreover, each field has defined and used a different kind of cover. While in the RDBM this base is the Canonical-Direct Basis or the Minimal Cover, the cover of choice in FCA is the so-called Duquenne-Guigues basis.

Both the implication problem and the problem of computing a cover are related: the implication problem is used in the algorithm **Canonical Basis** (Algorithm 16, page 103 in [14]) to compute the Duquenne-Guigues basis, and it is also used in the algorithm **Direct** (Section 5.4, Chapter 5 in [20]) which is used to compute the Minimal Cover. Again, the transversality of the Armstrong dependencies appears in a general concept (computing a cover) but in different forms (Duquenne-Guigues basis and Minimal Cover).

The purpose of this study is to present in a single paper the main different covers used in the RDBM, in Logic, and in FCA, and to discuss two closely related questions: (i) the computation of the closure of a set of attributes w.r.t. a set of Armstrong dependencies, (ii) the representation of such a set of the dependencies in a cover. To do so, we first present both Closure and Linclosure, debate their complexity and the different factors that may affect their performance. Afterwards, we review three different main covers that appeared in the literature. To the best of our knowledge, this is the first time that such a comprehensive study is proposed, where the connections between the relational database model, Logic, and FCA are examined from the implication problem perspective.

The paper is organized as follows. Section 2 presents the main aspects of the implication problem in the RDBM, Logic, and FCA. Then Section 3 provides the necessary definitions needed in the paper. Section 4 makes precise the Closure and Linclosure algorithms, while Section 5 includes a detailed comparison of the main covers. Finally, Section 6 concludes the paper and proposes an extensive discussion about these different and important covers.

2 The Relevance of the Implication Problem

In this section we briefly explain why the implication problem, that is, to check whether a dependency σ can be derived from a set of dependencies Σ by the iterative application of the Armstrong axioms is of importance in RDBM, logic, and FCA. We start by quoting the survey [13] which comes from the Relational Database field:

> *Most of the papers in dependency theory deal exclusively with various aspects of the implication problem, that is, the problem of deciding for a given set of dependencies Σ and a dependency σ whether Σ logically implies σ. The reason for the prominence of this problem is that an algorithm for testing implication of dependencies enables us to test whether two given sets of dependencies are equivalent or whether a given set of dependencies is redundant. A solution for the last two problems seems a significant step towards automated database schema design, which some researchers see as the ultimate goal for research in dependency theory.*

In what way are those two problems *a significant step towards automated database schema design* and what is the reason of the *prominence of* the implication problem? Functional dependencies are used to design relational schemes (see for example the extensive chapter 12.2 *Nonloss decomposition and functional dependencies* in [10]). However, *some of the FDs that are used to describe the data base may be redundant in the sense that they are derivable from the others.* These (redundant) FDs used to assist this method of schema design *will induce redundancy in the relational schema*, which is to be avoided since *redundancy in relations creates serious problems in maintaining a consistent data base.* Hence, one needs to be able to compute a *covering* (here we will use the term *cover*) of a set of dependencies, this is, a non-redundant set of FDs that yields the same closure with respect to the axioms for FDs. And, *the problem of finding a covering reduces to computing the predicate $\sigma \in \Sigma^+$, that is, $\Sigma \models \sigma$* (all portions of text in *italic* are citations from [6]).

Summing up, the implication problem for FDs is of importance because it solves the problem of computing a cover of a set of FDs which, in turn, prevents the propagation of redundancy in the design of a database scheme using functional dependencies.

In FCA, the basic data structure is the binary context which can have two related representations, namely the concept lattice and the basis of implications. The latter is the Duquenne-Guigues basis which is unique, minimal, and non redundant (see 5). There is an equivalence between these three views of an initial dataset. Moreover, one can be also interested in the so-called equivalence classes which are associated with one closed set and possibly several generators of different types [21]. A typical implication is related to any equivalence class which is of the form $X \rightarrow Y$ where Y is a closed set and closure$(X) = Y$. Then, a minimal basis has all its importance since it provides a summary of the dataset under study with a minimum number of elements. In particular, the number of implications in say the Duquenne-Guigues basis is much smaller than the number of concepts, that is, a substantial number of implications can be inferred from this minimum basis. This is also of importance when the problem of construction or reconstruction is considered, that is, starting with a set of implications, design the whole related dataset. This problem is present in the RDBM, in Logic and especially in the design of a theory or of an ontology, and as well in FCA with the attribute exploration process [14].

3 Definitions

In this section we introduce the definitions used in this paper. We do not provide the references for all of them because they can be found in all the textbooks and papers related to the RDBM, Logic and FCA.

As explained in the introduction, implications[15], functional dependencies [20] and Horn clauses [16] are dependencies between sets of attributes, which are equivalent from a syntactical point of view, since they are in agreement with the Armstrong axioms.

Definition 1. *Given set of attributes $\mathcal{U}$, for any $X, Y, Z \subseteq \mathcal{U}$, the Armstrong axioms are:*

1. ***Reflexivity**: If $Y \subseteq X$, then $X \to Y$ holds.*
2. ***Augmentation**. If $X \to Y$ holds, then $XZ \to YZ$ holds.*
3. ***Transitivity**. If $X \to Y$ and $Y \to Z$ hold, then $X \to Z$ holds.*

When we write that a dependency $X \to Y$ **holds**, we mean all the instances in which this dependency is valid or true. Therefore, the sentence *"If $X \to Y$ holds, then $XZ \to YZ$ holds"* can be rephrased as *"In any instance in which $X \to Y$ is valid, the dependency $XZ \to YZ$ is valid as well"*.

The Armstrong axioms allow us to define the closure of a set of dependencies as the iterative application of these axioms over a set of dependencies.

Definition 2. *Σ^+ denotes the closure of a set of dependencies Σ, and can be constructed thanks to the iterative application of the Armstrong axioms over Σ. This iterative application terminates when no new dependency can be added, and it is finite. Therefore, Σ^+ contains the largest set of dependencies that hold in all instances in which all the dependencies in Σ hold.*

The closure of a set of dependencies induces the definition of the cover of such a set of dependencies.

Definition 3. *The **cover** or **basis** of a set of dependencies Σ is any set Σ' such that $\Sigma'^+ = \Sigma^+$.*

We define now the concept of a *closure* of a set of attributes $X \subseteq \mathcal{U}$ with respect to a set of implications Σ.

Definition 4. *The closure of X with respect to a set of dependencies Σ is*

$$\text{closure}_\Sigma(X) = \{\, Y \mid X \to Y \in \Sigma^+ \,\}$$

that is, $\text{closure}_\Sigma(X)$ *is the largest set of attributes Y such that $X \to Y$ can be derived by the iterative application of the Armstrong axioms over the set Σ.*

This closure operation returns the largest set of attributes such that $\Sigma \models X \to \text{closure}_\Sigma(X)$. Therefore, the implication problem $\Sigma \models X \to Y$ boils down to testing whether $Y \subseteq \text{closure}_\Sigma(X)$.

4 Algorithms to Compute the Closure of a Set of Attributes

Below we review two most well-known algorithms to compute closure_Σ.

4.1 The Closure Algorithm

The first algorithm Closure appears in most of the main database textbooks, e.g., [1,20,27], and also in logic, where it is called **forward chaining** [9]. In Formal Concept Analysis the same algorithm that first appeared in the RDBM is reused as for example in [14].

Function Closure(X, Σ)

Input : A set of attributes $X \subseteq \mathcal{U}$ and a set of implications Σ
Output: $\text{closure}_\Sigma(X)$

```
1  stable ← false
2  while not stable do                                   // Outer loop
3  |  stable ← true
4  |  forall A → B ∈ Σ do                                // Inner loop
5  |  |  if A ⊆ X then
6  |  |  |  X ← X ∪ B
7  |  |  |  stable ← false
8  |  |  |  Σ ← Σ \ { A → B }
9  |  |  end
10 |  end
11 end
12 return X
```

In Table 1 we list different complexities of Closure as they are given in a classic database textbook [20], a FCA textbook [14] and the pioneer paper of Linclosure [4][5]. In general terms, the complexity of Closure depends on its two loops which are marked in the code as `outer` and `inner` loops. This algorithm iterates in the outer loop as long as there is a change in the computation of $\text{closure}_\Sigma(X)$. This means that in the worst case, the closure may be incremented by only one single attribute at each iteration of the outer loop, which implies that the outer loop is of order $\mathcal{O}(|\mathcal{U}|)$. Regarding the inner loop, it necessarily iterates over all the dependencies that are in Σ, that is, it is of order $\mathcal{O}(|\Sigma|)$. Then the total number of iterations, in the worst case, is of order $\mathcal{O}(|\mathcal{U}| \times |\Sigma|)$. Since in some cases it may happen that $|\Sigma| = |\mathcal{U}|$, the complexity of this algorithm can be, in the worst case, of order $\mathcal{O}(|\Sigma|^2)$.

[5] Other relevant textbooks on RDBM line [1,27] provide the same complexity analysis as in Table 1

However, there are two extra comments about this complexity analysis. First, we have stated that the inner loop (line 4) iterates over all the dependencies in Σ, but all the dependencies that have been used to compute closure$_\Sigma(X)$ are deleted in line 8. But even if this removal is performed, the number of iterations is still of order $|\Sigma|$, since in the worst case, we may remove only one single dependency from Σ at each iteration of the inner loop. Also, in line 5, there is a subset containment check of order $\mathcal{O}(|\mathcal{U}|)$ that is performed as many times as there are iterations of the inner loop. This fact, which is only considered in [20], induces a complexity of order $\mathcal{O}(|\mathcal{U}| \times |\Sigma|^2)$. As we can observe in Table 1, the consensus is that the complexity of Closure is of order $\mathcal{O}(|\Sigma|^2)$ (in the worst case scenario $|\mathcal{U}| = |\Sigma|$). However, in cases when $|\mathcal{U}| << |\Sigma|$, the complexity of Closure becomes of order $\mathcal{O}(|\mathcal{U}| \times |\Sigma|)$. One of these examples can be found in Section 3.2.3 in [14], where $|\Sigma|$ is exponential w.r.t. $|\mathcal{U}|$[6].

In any case, Closure is considered a **quadratic** time algorithm with respect to the number of dependencies.

Ref	Inner	Outer	Total
Maier [20]	$\|\mathcal{U}\| \times \|\Sigma\|$	$\|\Sigma\|$	$\|\mathcal{U}\| \times \|\Sigma\|^2$
Ganter-Obiedkov [14]	$\|\Sigma\|$	$min(\|\mathcal{U}\|, \|\Sigma\|)$	$min(\|\mathcal{U}\| \times \|\Sigma\|, \|\Sigma\|^2)$
Beeri-Bernstein [4]	$\|\Sigma\|$	$\|\mathcal{U}\|$	$\|\mathcal{U}\| \times \|\Sigma\|$

Table 1. Complexity of the Closure algorithm according to different authors.

4.2 The Linclosure Algorithm

An improved version of Closure is the Linclosure algorithm [4]. In particular, Linclosure improves Closure by annotating in the preparation part the dependencies which do not need to be checked to compute closure$_\Sigma(X)$. It should be noticed that the "chain forward algorithm" of Marcel Wild [28] is in fact an improvement of Linclosure which is not covered in this paper.

Linclosure consists of two parts: a `preparation` part in which the necessary data structures are computed, and the `computation` part in which the computation of closure$_\Sigma(X)$ is performed. In the first part two data structures are designed: one structure contains, for each attribute say Z, a pointer to all the dependencies $X \to Y$ such that Z appears in the left-hand side X. The second structure is such that for each dependency $X \to Y$ there is a counter that annotates the number of attributes of its left-hand side X which have already been visited while computing closure$_\Sigma(X)$. The objective of these two structures is to only visit the dependencies that are necessary to compute the closure, and to ignore those that are not useful to compute it. Regarding the complexity of

[6] *"the number of pseudo-closed sets [Σ] can be exponential in comparison to the size of the base set [U]"*

Function LINCLOSURE(X, Σ)

Input : A set of attributes $X \subseteq \mathcal{U}$ and a set of implications Σ
Output: closure$_\Sigma(X)$

```
1  forall A → B ∈ Σ do                                          // Preparation
2  |  count[A → B] ←| A |
3  |  if | A |= 0 then
4  |  |  X ← X ∪ B
5  |  end
6  |  forall a ∈ A do
7  |  |  list[a] ← list[a] ∪ { A → B }
8  |  end
9  end
10 update ← X
11 while update ≠ ∅ do                                          // Outer loop
12 |  choose m ∈ update
13 |  update ← update \ { m }
14 |  forall A → B ∈ list[m] do                                 // Inner loop
15 |  |  count[A → B] ← count[A → B] − 1
16 |  |  if count[A → B] = 0 then
17 |  |  |  add ← B \ X
18 |  |  |  X ← X ∪ add
19 |  |  |  update ← update ∪ add
20 |  |  end
21 |  end
22 end
23 return X
```

Linclosure, there is a general consensus that this algorithm is of order $\mathcal{O}(|\Sigma|)$ (here we do not discuss the complexity of the preparation part, which is assumed to be of the same complexity as the rest of the algorithm). One explanation of this fact appears in the pioneering paper [4], page 47 in the second paragraph[7]:

> *For each attribute in [update], the [outer] loop follows a constant number of steps for each occurrence of that attribute on the left side of an FD in Σ. Similarly, each right side of an FD in Σ is visited at most once in [the outer loop]. Thus [the outer loop] is also $\mathcal{O}(|\Sigma|)$ as is the entire Algorithm*[8]*.*

Two comments should be made here. The first one is that the sentence *"each right side of an FD in Σ is visited at most once"* also applies to Closure (lines 6

[7] We adapt this paragraph to fit names in Algorithm Linclosure.
[8] Previously, the authors have concluded that the complexity of the preparation part is of order $\mathcal{O}(|\Sigma|)$, as well as the second part, hence *"is also $\mathcal{O}(|\Sigma|)$ the entire Algorithm"*.

and 7) since a dependency is removed once it has been used (line 8 of Closure). The second one is that *"For each attribute in [update], the [outer] loop follows a constant number of steps for each occurrence of that attribute on the left side of an FD in Σ"* tells us that for each attribute in *update* (which is the variable that contains the attributes that still need to be processed in the outer loop) there is a **constant** number of steps that are performed in each left hand side **of each dependency that contains that attribute** (in its left-hand side). But this number of dependencies is of order $\mathcal{O}(|\Sigma|)$.

From our understanding, this is relevant since the total number of iterations that are performed in Linclosure is the same as the number of times that the algorithm executes lines 15 and 16, and this number may be, in the worst case, when **all** the dependencies are decremented as many times as the number of attributes in their left-hand side, that is, of order $\mathcal{O}(|\mathcal{U}| \times |\Sigma|)$. And here we have a caveat: the complexity of both Closure and Linclosure is given with respect to the **size of the input**, that is, the size of Σ. However, in some cases this size is interpreted as the number of dependencies that are in Σ, that is, $|\Sigma|$, and in some other cases, it is interpreted as the size in number of symbols, that is, $|\mathcal{U}| \times |\Sigma|$. For example, in [20], the complexity of Linclosure is of order $\mathcal{O}(n)$ for an input length n, where $n = |\mathcal{U}| \times |\Sigma|$, whereas in [4] this complexity, is of order $\mathcal{O}(|\Sigma|)$ as we have stated.

4.3 Experimental Evaluation of Closure and Linclosure

As mentioned above, the general consensus is that the complexity order of Closure is quadratic in the worst case, whereas Linclosure is assumed to be of linear cost. But several simple examples (see [3], for example) show that the performance of both algorithms is sensitive to different factors, e.g., the size of Σ or the order in which Σ is explored. Moreover, the expected performance of Closure and Linclosure did not fully match their empirical evaluation in [3]. To be more precise, the experiments were not fully conclusive in supporting the theoretical complexity analysis, since in some scenarios Closure unexpectedly outperformed Linclosure. In other words, Linclosure did not clearly outperform Closure in all scenarios as expected.

Actually, we need to take into account that closure_Σ has two parameters, namely a set of dependencies Σ and a set of attributes X, and the complexity of the algorithms is given with respect to the size of both sets. In this case, the only parameter that admits some degree of variation is Σ. Therefore, there is one parameter that is critical in the computation of closure_Σ, that is, the size of Σ, as we will make it precise later.

5 Covers of Dependencies

We now present the different types of covers that are present and adopted in the three fields, namely RDBM, FCA, and Logic. Since the definition of a cover (Definition 3) is very generic, the covers reviewed here have been defined with

respect to specific characteristics and for different purposes. As seen above, one of the parameters of both Closure and Linclosure algorithms is a set of dependencies Σ. Then the nature and the size of Σ affect the performance of both functions. On the other hand, the algorithms that compute those covers make use of a function to compute the closure of a set of attributes, for example both functions Closure and Linclosure can be used. Therefore, the implication problem or its equivalent $\text{closure}_\Sigma(X)$ and the computation of a cover are two interrelated problems.

5.1 Four Main Characteristics of Covers

In general terms, a cover is simply a set of dependencies Σ. The definition of a cover is very generic and below we introduce some relevant properties which are useful for characterizing the different covers. In particular, Σ is equivalent to another set of dependencies Σ' modulo the Armstrong axioms when the closures Σ^+ and Σ'^+ are the same.

Definition 5. *A set of dependencies Σ is **left-reduced** if and only if for all $X \to Y \in \Sigma$ there is no $X' \to Y$, where $X' \subset X$, such that changing $X \to Y$ by $X' \to Y$ in Σ gives an equivalent base.*

Alternatively, the definition states that $((\Sigma \setminus \{X \to Y\}) \cup \{X' \to Y\})^+ \neq \Sigma^+$ for all $X \to Y \in \Sigma$. This means that all the left-hand sides of all the dependencies in Σ cannot be further reduced to compute the closure Σ^+.

The process of left-reducing a set of dependencies is also mentioned as the removal of **extraneous attributes** in the left-hand sides of all dependencies. As it can be expected, an attribute x is extraneous in the left-hand side of a dependency $\sigma \in \Sigma$ if removing x from the left-hand side in σ does not change Σ^+. The dual definition can be given. An attribute y is extraneous in the right-hand side of a dependency $\sigma \in \Sigma$ if removing y from the right-hand side in σ does not change Σ^+.

Definition 6. *A set of dependencies Σ is **non redundant** if and only if $(\Sigma \setminus \sigma)^+ \neq \Sigma^+$ for all $\sigma \in \Sigma$.*

If a set of dependencies Σ is non redundant and all the right-hand sides are singletons then Σ is **right-reduced**. In fact, non-redundancy and right-reduction are equivalent definitions, as Example 1 shows.

Example 1. Let $\Sigma = \{ a \to bc, b \to c, a \to d \}$ be a set of dependencies. The attribute c in the dependency $a \to bc$ is clearly extraneous, since the base $\Sigma' = \{ a \to b, b \to c, a \to d \}$ is equivalent to Σ. The transformation of Σ into Σ' can be viewed from two different perspectives: we remove the attribute c in the dependency $a \to bc$ because it is extraneous (right-reduction) or we take the base $\Sigma = \{ a \to b, a \to c, b \to c, a \to d \}$ (in which all the right-hand sides are singletons) from which we remove the dependency $a \to c$ because it is redundant (non-redundancy).

Here we keep the expression "non-redundant" instead of "right-reduced" for the sake of clarity. Since $\{ X \to yz \} \equiv \{ X \to y, X \to z \}$, there is no difference between considering a cover such that all the right-hand sides are singletons, or just *joining* in one single dependency all those dependencies with the same left-hand side. Actually, when the right-hand sides are singletons, the number of dependencies is "artificially" increased.

Definition 7. *A set of dependencies Σ is **direct** if and only if* $\text{closure}_\Sigma(X)$ *can be computed with a single pass of Σ, for all $X \subseteq \mathcal{U}$.*

This means that computing the closure of any $X \subseteq \mathcal{U}$ implies only one complete exploration of Σ. Actually, this unique exploration represents a very important property, especially when Σ is large or very large. Moreover, it should also be noticed that (i) a cover Σ is direct regardless of how it is represented or sorted, and also (ii) Σ is direct for any attribute subsets $X \subseteq \mathcal{U}$.

Definition 8. *A set of dependencies Σ is **minimal** if and only if $|\Sigma| \leq |\Sigma'|$ for all Σ' such that $\Sigma^+ = \Sigma'^+$.*

It is important to notice that when a cover is minimal, it is also non redundant, but the opposite does not necessarily hold. Moreover, let us recall also the importance of computing a non redundant cover given a set of dependencies Σ, and that computing a non redundant cover is equivalent to the logical implication problem.

In the next sections we review three different and major types of covers that are of interest in the RDBM, in Logic, and in FCA.

5.2 The Minimal Cover in the RDBM and its variations

We start with the **Minimal Cover**, which is very popular among the RDBM community and can be found in most of the database textbooks under different names (see Table 2).

Name	Ref	Where
Canonical Cover	Maier [20]	p. 79, Section 5.6
Minimal Cover	Ullman [27]	p. 390
Minimal Cover	Abiteboul [1]	p. 286, Exercice 11.16
Irreducible Set of Dependencies	Date [10]	p. 341, Section 11.6
Minimal Cover	Elmasri [12]	p. 549, Section 16.1.3
Canonical Cover	Silberschatz [26]	p. 324, Section 7.4.3

Table 2. References to the Minimal Cover in RDBM textbooks.

The computation of the Minimal Cover is performed in three steps:

1. All the dependencies in Σ must have only one single attribute in the right-hand side. This is performed by simply replacing a dependency $X \to Y$ by the dependencies $X \to y_i$ for all $y_i \in Y$.
2. Σ is left-reduced. This is performed by changing a dependency $X \to y$ by a dependency $X' \to y$, where $X' \subset X$, whenever $(\Sigma \setminus \{X \to Y\} \cup \{X' \to Y\})^+ \equiv \Sigma^+$ (see Definition 5).
3. Redundant dependencies are removed from Σ (see Definition 6).

It is important to notice that the order of steps 2 and 3 is relevant and mandatory. Section 5.3 in [20] includes a discussion explaining why left-reduction needs to be performed before the removal of redundant dependencies. The output of computing the Minimal Cover depends also on the order in which dependencies are processed. As a consequence it comes that there may be different minimal covers for the same Σ. Finally, the Minimal Cover does not ensure directness.

Example 2 (Adapted from Section 5.2 in [20]).

Let us suppose that we have the following set of dependencies: $\Sigma = \{ a \to b, b \to ac, a \to c, bc \to a \}$. Applying step 1 outputs $\Sigma = \{ a \to b, b \to a, b \to c, a \to c, bc \to a \}$. Then step 2 is applied to left-reduce Σ. Since $bc \to a$ can be left-reduced to $b \to a$ (thanks to Augmentation), then the following equivalent set is produced: $\Sigma' = \{ a \to b, b \to a, b \to c, a \to c \}$. Finally, applying step 3 outputs: $\Sigma'' = \{ a \to bc, b \to a \}$. By contrast, let us assume that the order of Σ is changed as follows: $\Sigma = \{ a \to b, a \to c, b \to ac, bc \to a \}$. Applying step 1 yields the same set as above: $\Sigma = \{ a \to b, a \to c, b \to a, b \to c, bc \to a \}$. When applying step 2, it comes: $\Sigma' = \{ a \to b, a \to c, b \to a, b \to c \}$. And finally applying step 3 outputs the base $\Sigma' = \{ a \to b, b \to ac \}$.

5.3 The Canonical-Direct Basis

The Canonical-Direct Basis is defined with different characterizations in [8]. Again the computation of this cover is performed in three steps:

1. All the dependencies in Σ must have one single attribute in the right-hand side, as it was already the case above for the computation of the minimal cover. Again this is performed by simply replacing a dependency $X \to Y$ by the dependencies $X \to y_i$ for all $y_i \in Y$.
2. Σ is closed by "pseudo-transitivity", that is, Augmentation plus transitivity. Then dependencies $X \to y$ such that $y \in X$ are removed.
3. Σ is left-reduced (see Definition 5 and step 2 in the construction of the minimal cover).

It can be noticed that there is no removal of redundant dependencies. The only source of redundancy that is taken into account and removed is the one generated by the application of augmentation, but not of transitivity. The Canonical-Direct Basis is not necessarily minimal, nor it is non-redundant, but it is direct.

Example 3. We continue with Example 2: $\Sigma = \{ a \to b, b \to ac, a \to c, bc \to a \}$. Applying step 1 produces $\Sigma = \{ a \to b, b \to a, b \to c, a \to c, bc \to a \}$. Here, step 2 consists in closing Σ by pseudo-transitivity, which outputs: $\Sigma' = \{ a \to b, b \to a, b \to c, a \to c, bc \to a, a \to a, ac \to a, ab \to a, ab \to b, ac \to c \}$. Then, removing the dependencies that can be deduced by Reflexivity, it comes: $\Sigma'' = \{ a \to b, b \to a, b \to c, a \to c, bc \to a \}$. When applying step 2 to left-reduce Σ, and since $bc \to a$ can be left-reduced to $b \to a$ (thanks to Augmentation), the following equivalent set is obtained and constitutes the final result: $\Sigma''' = \{ a \to b, b \to a, b \to c, a \to c \}$, Since there is no removal of redundant dependencies, then dependency $a \to c$ appears in this canonical-direct basis. By contrast this was not the case in the minimal basis above where $a \to c$ was removed (see Example 2).

We need to notice that this base is also characterized in Formal Concept Analysis by the so called **minimal generators** (or minimal generating set). A minimal generator is defined in [14] (Section 2.3.3) as follows: *A generating set of a closed set A is a subset $S \subseteq A$ such that $A = S''$, and, obviously, a minimal generating set of A is a subset of A minimal with respect to this property.* A similar definition exists in [7]: *A minimal generator B of a closed set F is also called a basis for F , i.e. $\phi(B) = F$ and $\phi(A) \subset \phi(B)$ for every $A \subset B$, or a free subset, i.e. for every $x \in B$, $x \in \phi(B \setminus x)$* (where ϕ is a closure operator). What is also relevant is that, in this same paper, the characterization of the canonical direct basis Σ_{cdb} is defined as follows: $\Sigma_{cdb} = \{ B \to \text{closure}_\Sigma(B) \setminus B : B \subseteq \mathcal{U}$ is a minimal generator of $\text{closure}_\Sigma(B) \}$. That is, the left-hand sides of a Canonical-Direct Basis are the set of minimal generators of the implicit closure operator.

5.4 The Duquenne-Guigues basis

The **Duquenne-Guigues basis** [11,15], also called the *Canonical Basis* in the FCA community, is the cover based on pseudo-closed sets [15]. More precisely, tt is defined as follows:

Definition 9. *The **Duquenne-Guigues basis** of a set of dependencies Σ is defined as*

$$\{ X \to \text{closure}_\Sigma(X) \mid X \subseteq \mathcal{U} \text{ and } X \text{ pseudoclosed} \}$$

where the definition of a pseudo-closed set of attributes w.r.t. a set of dependencies Σ is:

Definition 10. *Let Σ be a set of dependencies, and $\mathcal{U}$ the related set of attributes. $X \subseteq \mathcal{U}$ is **pseudoclosed** if:*

1. *$X \neq \text{closure}_\Sigma(X)$, that is, X is not closed.*
2. *If $Y \subset X$ is a proper subset of X and pseudo-closed, then $\text{closure}_\Sigma(Y) \subseteq X$.*

This basis is not direct, but it is minimal and non-redundant. According to [8], this basis is also presented by Maier in [20], where it is called the *Minimum Cover*: *"It has been obtained independently (and with different formulations) by Maier (Minimum Cover), and Guigues and Duquenne (Duquenne-Guigues basis)"*. However, at present, the use and popularity of the Duquenne-Guigues basis seems to be rather restricted within the FCA community [14,15].

Example 4. Let us consider Example 2: $\Sigma = \{ a \to b, b \to ac, a \to c, bc \to a \}$.

As in Σ the pseudo-closed sets are simply $\{ a \}$ and $\{ b \}$, the following Duquenne-Guigues basis is obtained: $\Sigma' = \{ a \to bc, b \to ac \}$.

6 Discussion and Conclusion

For being more complete, we mention without giving more details another basis, namely the **D-Basis** [2], which can be considered as being *between* the Canonical-Direct Basis and the Minimal Cover. This particular basis, developed in lattice theory, while not minimal, is direct. Moreover, it is smaller in size than the canonical-direct basis, that is, D-Basis $\subseteq$ Canonical-Direct Basis.

Relating the three main covers plus the D-Basis is relevant and very interesting. Actually, while the D-Basis and the Canonical-Direct Basis are related by a subset relationship, such a relationship is not known to exist between the Duquenne-Guigues basis and the Minimum Cover, or between the Canonical-Direct Basis and the Duquenne-Guigues basis. In fact, in [8], in the last sentence before the acknowledgements page 28, it is stated that:

> *We conclude that this paper is contradicting a conjecture of the literature (in [37]). Indeed, one observes that the premise of implication (10) of Σ_{cd} (a direct cover) is not contained in a premise of any implication of Σ_{can} (the Duquenne-Guigues basis).*

This sentence goes back to a conjecture stated in [5] and can be interpreted as follows. The left-hand sides of a Duquenne-Guigues basis, which is minimal, may not be a subset of the left-hand sides of a direct cover.

The RDBM, Logic, and FCA fields are addressing two different, yet related problems: the implication problem and the computation of a compact and representative set –cover or basis– of a complete set of dependencies. Although the first question is solved in the three fields thanks to the same algorithms, that is, Closure or Linclosure, this unanimity disappears in confronting with the choice of a cover. Table 3 summarizes the three types of covers reviewed in Section 5, plus the D-Basis. It can be noticed that the Canonical-Direct Basis and the D-Basis do not keep dependencies that can be inferred by the application of the augmentation axiom: $X \to Y \models XZ \to YZ$, while they include dependencies that can be inferred by transitivity. This additional amount of information is enough to make direct these two covers. By contrast, the Minimal Cover does not contain dependencies that can be inferred by augmentation or transitivity

	Canonical Minimal	Canonical Direct	Duquenne-Guigues Minimum	D-Basis
(found in)	RDBM	RDBM, Logic	FCA/RDBM	Lattice Th.
Minimum	no	no	yes	no
Direct	no	yes	no	yes
Redundant	no	yes	no	yes
Unique	no	yes	yes	yes

Table 3. Comparing the characteristics of the four bases.

as they are removed in the last step of the computation. And this explains why the Minimal Cover is not direct.

We have there a kind of "no free lunch theorem": the more information a basis is keeping the more direct the basis can be. By contrast, minimality and non redundancy do not favor directness.

The lack of unanimity can also be noticed in the RDBM only. Indeed textbooks such as [1,20,27,10,12,26] tend to present the Minimal Cover as the preferred cover, while algorithms computing FDs output the Canonical-Direct Basis [24]. This can be interpreted as follows: textbooks are preferring a cover left-reduced and non-redundant, and, hence, containing less and more compact information. However, for discovering FDs, a left-reduced cover is computed without removing redundant dependencies. A possible explanation is that algorithms computing FDs take a dataset as input. Then it is easy to perform a left-reduction w.r.t. the input dataset by removing an attribute from the left-hand side and test whether the dependency still holds. However, a redundancy test is different in the sense that it can only be performed w.r.t. a set of dependencies, *once this set has completely been computed*. Such a test is of a different nature as it is not performed w.r.t. a dataset. Moreover, this does not prevent any algorithm from performing it.

Although the Minimum Cover (or Duquenne-Guigues basis) is also introduced in the RDBM field, it has not enjoyed the same popularity as the Minimal Cover. In fact, the Minimum Cover is introduced and discussed only in Maier [20]. This lack of popularity is probably due to the rather intricate characterization of the Minimum Cover and the related algorithm at that time (see for example Section 5.6, Chapter 5 in [20]). This is especially true when compared with the simplicity and expressiveness of the presentation of the Minimal Cover. The characterization of Minimum Cover cannot compete either with the clear characterization of the Duquenne-Guigues basis in FCA, or the simplicity of the NextClosure Algorithm in [14]. However, this "simplicity" is not for free and it comes at the expense of the whole theoretical framework of FCA.

The choice of a cover can be decided depending on the performance that Closure or Linclosure are offering. Both Closure and Linclosure are closely dependent on the "nature" of the set of dependencies Σ. By "nature" we mean the different characteristics explained in Section 5, which have an impact on the

amount of information as well as on the number of dependencies considered. If Σ is a direct cover (Definition 7), both algorithms have to perform only a single pass of their outer loop. If the cover is not direct, e.g., Canonical-Direct Basis or Duquenne-Guigues basis, then, the number of iterations of the outer loop may be larger, while, at the same time, the number of iterations of the inner loop may be shorter. Again, we are facing the well-known trade-off between "expressivity and complexity": the more expressive in terms of information containment a cover is, the higher the cost of Closure and Linclosure is.

The question of the preference between the Duquenne-Guigues basis and Minimum Cover remains, even if things have changed. What is left for future work is in concern with the practical behavior of the main covers, which has to be compared and investigated from the experimental point of view, especially having in mind the needs in the RDBM, in Logic, and in FCA, but also in knowledge representation and ontology engineering. In particular, attribute exploration [14] may be a good support for evaluating the potential of the main covers studied above in ontology engineering and database design.

7 Acknowledgements

J. Baixeries is funded by a grant AGRUPS-2022 from Universitat Politècnica de Catalunya and is supported by a recognition 2021SGR-Cat (01266 LQMC) from AGAUR (Generalitat de Catalunya). Mehdi Kaytoue and Amedeo Napoli are carrying out this research work as part of the French ANR-21-CE23-0023 SmartFCA Research Project.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Adaricheva, K.V., Nation, J.B., Rand, R.: Ordered direct implicational basis of a finite closure system. Discret. Appl. Math. **161**(6), 707–723 (2013)
3. Bazhanov, K., Obiedkov, S.A.: Optimizations in computing the Duquenne-Guigues basis of implications. Ann. Math. Artif. Intell. **70**(1-2), 5–24 (2014)
4. Beeri, C., Bernstein, P.A.: Computational problems related to the design of normal form relational schemas. ACM Trans. Database Syst. **4**(1), 30–59 (1979)
5. Ben-Khalifa, K., Motameny, S.: Horn representation of a concept lattice. Int. J. Gen. Syst. **38**(4), 469–483 (2009)
6. Bernstein, P.A., Swenson, J.R., Tsichritzis, D.C.: A Unified Approach to Functional Dependencies and Relations. In: Proceedings of ACM SIGMOD. pp. 237–245 (1975)
7. Bertet, K., Demko, C., Viaud, J.F., Guérin, C.: Lattices, closures systems and implication bases: A survey of structural aspects and algorithms. Theoretical Computer Science **743** (11 2016)
8. Bertet, K., Monjardet, B.: The multiple facets of the canonical direct unit implicational basis. Theor. Comput. Sci. **411**(22-24), 2155–2166 (2010)

9. Crama, Y., Hammer, P.L.: Boolean Functions - Theory, Algorithms, and Applications, Encyclopedia of mathematics and its applications, vol. 142. Cambridge University Press (2011)
10. Date, C.J.: An introduction to database systems (7. ed.). Addison-Wesley-Longman (2000)
11. Duquenne, V., Guigues, J.: Mininal family of informative implications resulting from a binary data table. Mathematics and Humanies Sciences **24**, 5–18 (01 1986)
12. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, 3rd Edition. Addison-Wesley-Longman (2000)
13. Fagin, R., Vardi, M.Y.: The Theory of Data Dependencies - An Overview. In: Proceedings of the 11th International Colloquium on Automata, Languages, and Programming. pp. 1–22. Lecture Notes in Computer Science 172, Springer (1984)
14. Ganter, B., Obiedkov, S.A.: Conceptual Exploration. Springer (2016)
15. Ganter, B., Wille, R.: Formal Concept Analysis - Mathematical Foundations. Springer (1999)
16. Horn, A.: On sentences which are true of direct unions of algebras. J. Symb. Log. **16**(1), 14–21 (1951)
17. Ibaraki, T., Kogan, A., Makino, K.: Functional dependencies in horn theories. Artif. Intell. **108**(1-2), 1–30 (1999)
18. Kanellakis, P.C.: Chapter 17 - Elements of Relational Database Theory. In: Van Leeuwen, J. (ed.) Formal Models and Semantics, pp. 1073–1156. Handbook of Theoretical Computer Science, Elsevier, Amsterdam (1990)
19. Kowalski, R.A.: Logic for problem solving, The computer science library : Artificial intelligence series, vol. 7. North-Holland (1979)
20. Maier, D.: The Theory of Relational Databases. Computer Science Press (1983)
21. Makhalova, T., Buzmakov, A.V., Kuznetsov, S.O., Napoli, A.: Introducing the closure structure and the GDPM algorithm for mining and understanding a tabular dataset. International Journal on Approximate Reasoning **145**, 75–90 (2022)
22. Mannila, H., Räihä, K.: Design of Relational Databases. Addison-Wesley (1992)
23. Padawitz, P.: Computing in Horn Clause Theories, EATCS Monographs on Theoretical Computer Science, vol. 16. Springer (1988)
24. Papenbrock, T., Ehrlich, J., Marten, J., Neubert, T., Rudolph, J.P., Schönberg, M., Zwiener, J., Naumann, F.: Functional dependency discovery: An experimental evaluation of seven algorithms. Proc. VLDB Endow. **8**(10), 1082–1093 (jun 2015)
25. Sagiv, Y., Delobel, C., Jr., D.S.P., Fagin, R.: An Equivalence Between Relational Database Dependencies and a Fragment of Propositional Logic. J. ACM **28**(3), 435–453 (1981)
26. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts, Seventh Edition. McGraw-Hill Book Company (2020)
27. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, Volume I, Principles of computer science series, vol. 14. Computer Science Press (1988)
28. Wild, M.: Computations with finite closure systems and implications. In: Du, D., Li, M. (eds.) Computing and Combinatorics, First Annual International Conference, COCOON '95, Xi'an, China, August 24-26, 1995, Proceedings. Lecture Notes in Computer Science, vol. 959, pp. 111–120. Springer (1995)