Letting Keys and Functional Dependencies out of the Bag

Sebastian Link **Mozhgan Memari**

Department of Computer Science,
The University of Auckland,
Auckland, New Zealand,
Email: [s.link|m.memari]@auckland.ac.nz

Abstract

Classical database theory is largely a theory of relations. Relations are sets of tuples in which no duplicate tuples occur. In practice, duplicate elimination is an operation that is considered to be too expensive in many situations. Fundamental classes of integrity constraints interact differently over bags than they do over relations. This holds for keys and functional dependencies, for example. In this paper, we establish algorithms that compute Armstrong bags for any given set of keys and functional dependencies. These are bags which satisfy the given set of keys and functional dependencies, but violate all keys and functional dependencies not implied by the given set. Armstrong bags perfectly visualize abstract sets of constraints, and can be used by database designers to effectively communicate perceptions or states of a database. Subsequently, we show how existing algorithms to discover functional dependencies in relations can be adapted to discover keys and functional dependencies in bags.

Keywords: Armstrong database, Bag, Constraint, Cover, Discovery, Functional dependency, Key

1 Introduction

A database system is a software package that manages a collection of persistent information in a shared, reliable, effective and efficient way. Most database systems are still founded on *the relational model of data* (Codd 1970). In this model, data is stored in a collection of relations that may vary over time. A relation is a set of tuples over a given time-invariant relation schema. The relation schema itself is a set of attributes that model all the properties that each tuple of each relation over the schema is described by. That is, a tuple maps each attribute of the relation schema to a value from the domain (a set of possible values) of this attribute. One may think of a relation as a table in which the column headers are given by the attributes of the relation schema, and each row of the table is given by a tuple.

This research is supported by the Marsden fund council from Government funding, administered by the Royal Society of New Zealand.

There is a gap between database theory and practice: all commercial relational database management systems allow duplicate tuples, and so relations in such systems are in effect bags. That is, duplicate tuples usually occur in real world database instances, even though the relational model of data is set-oriented, i.e., does not allow duplicate tuples to occur. Indeed, databases and bags are tightly coupled. For instance, the cost of duplicate removal, e.g. after projections of relations or the union of several relations, is frequently considered to be too expensive. Therefore, duplicate removal is only performed if the user explicitly requests so. Duplicate detection has evolved into its own area of research in recent years (Naumann & Herschel 2010). Furthermore, bag processing has applications in the evaluation of database queries, and in areas such as view maintenance, data warehousing and web information discovery. Consequently, there has been an effort in database research to establish a foundation for handling bags instead of sets, cf. (Lamperti et al. 2000).

1.1 Background and Motivation

Surprisingly, an extension of the relational framework of *data dependencies* to bags has not received much attention. Data dependencies are specified as semantic constraints on a relation schema that restrict the set of possible database instances to those which are considered meaningful to the application domain at hand. The intuitive meaning of "dependency" is that the occurrence of data values satisfying a set of certain properties enforces some properties of other data values. In this sense, the latter values are "dependent" on the former ones. A prime example is given by the class of *functional dependencies* (FDs). These are expressions of the form $X \rightarrow Y$ where X and Y are subsets of the same relation schema R. An instance over R satisfies this functional dependency whenever any two tuples of the instance agree on all attributes of X, then they also agree on all the attributes of Y. In this sense, the values on attributes of Y are *functionally dependent* on the values on attributes of X. FDs may *imply* other FDs, i.e., whenever a relation satisfies a set of FDs, then it may also satisfy other FDs. The *implication problem* of FDs is to decide for an arbitrary relation schema and an arbitrary set $\Sigma \cup \{\varphi\}$ of FDs on the relation schema, whether Σ implies φ. The finite axiomatisability, i.e. the existence of a finite, sound and complete set of syntactical inference rules, and the time-complexity of the implication problem for FDs in relational databases has been well-studied in the literature (Armstrong 1974, Diederich & Milton 1988). In particular, the implication of FDs is equivalent to the implication of Horn clauses in Boolean propositional logic (Fagin 1982*b*).

$\frac{}{kR}$ (set axiom)	$\frac{kX}{kXY}$ (superkey)
Key Axiomatisation	

$\frac{}{XY \to X}$ (reflexivity)	$\frac{X \to Y}{X \to XY}$ (extension)	$\frac{X \to Y \quad Y \to Z}{X \to Z}$ (transitivity)
Armstrong Axioms for FDs		

Table 1: Axiomatizations of Keys, and of FDs in Relations

Since Horn clauses are extremely important in computer science, so are FDs. In database practice, in particular, they are fundamental to data modeling, data cleaning, query optimization, database security and schema design (Thalheim 2000). FDs play an important role in other data models as well (Arenas & Libkin 2005, Hartmann & Link 2012, Vincent et al. 2004, Weddell 1992). In schema design it is a desirable goal to avoid data redundancy in future database instances. The absence of data redundancy results in databases that can be updated efficiently (Vincent 1999), and in which inference control reduces to access control (Biskup & Lochner 2008). A relation schema R is usually considered *well-designed* with respect to the set Σ of FDs specified on R, if for every $X \to Y \in \Sigma$ the set X either contains all the attributes of Y, or $X \to R$ is implied by Σ (Biskup et al. 1979). Indeed, a relation satisfies the functional dependency $X \to R$ precisely when it satisfies the *key* kX on R. That is, no two distinct elements of the relation over the schema R may have matching values on all the attributes in X.

The equivalence between a key kX and an FD $X \to R$ becomes invalid, if bags are permitted as database instances over the schema R. Bags can contain duplicate tuples, which have matching values on all the attributes of R. Indeed, a bag may satisfy the functional dependency $X \to R$, but some distinct tuples of the bag may still agree on all the attributes in X, i.e., violate the key kX.

Example 1 *Consider a database where we store customer orders, e.g. for Japanese cuisine. We record the* Item*, and the* Size *and* Price *of each item. An order may look as follows:*

Item	*Size*	*Price*
takoyaki	*large*	*¥90*
takoyaki	*large*	*¥90*
nigiri sake	*large*	*¥90*
nigiri sake	*average*	*¥70*
nigiri sake	*small*	*¥70*
okonomiyaki	*small*	*¥300*

For this bag, no subset of {Item,Size,Price} is a key. In particular, it does not satisfy the key k{Item,Size}. However, it does satisfy the functional dependency Item,Size → Item,Size,Price.

Consequently, it is a natural question to ask how the implication problem of keys and FDs can be characterized when bags are permitted as database instances. The question is significant since relational database management systems permit bags of tuples, but the underlying relational model of data only permits sets of tuples. In previous work (Köhler & Link 2010) we have characterized the implication problem axiomatically, algorithmically, and logically. For example, the inference rules from Table 2 form a minimal axiomatization (Köhler & Link 2010). In particular, the superkey rule follows from reflexivity axiom and the pullback rule. Logically, FDs correspond to definite Horn clauses, and keys to goal Horn clauses; refining the correspondence between FDs and Horn clauses over relations (Köhler & Link 2010).

1.2 Contributions

In this paper, we are interested in the discovery of keys and functional dependencies. First, we will address how to support the effective discovery of keys and functional dependencies that are semantically meaningful for a given application domain. In practice, there is a mismatch in expertise. Domain experts know a lot about a given application domain, but do not know anything about database concepts. Database designers know a lot about database concepts, but do not know a lot about the application domain for which they are hired to develop an information system. This mismatch makes it difficult to exchange perceptions about the application domain. In particular, database designers and business analysts face the difficult task to acquire the constraints perceived semantically meaningful by the domain experts. The task is difficult since database constraints are quite abstract and their interaction is even harder to communicate. Since humans can learn a lot from examples, data samples that faithfully represent abstract sets of constraints could be a helpful tool for the discovery of semantically meaningful constraints. The notion of data samples that faithfully represent abstract constraint sets is known as *Armstrong samples* in the database literature (Fagin 1982*a*, Link 2012). A data sample is Armstrong for a given set Σ of constraints in a given class $\mathcal{C}$, if the data sample satisfies all constraints in Σ and violates all constraints in $\mathcal{C}$ that are not implied by Σ. Therefore, if Σ represents the set of constraints currently perceived meaningful by a design team, then an Armstrong sample for Σ satisfies all constraints currently perceived meaningful, and violates all constraints currently perceived meaningless. The simple and fundamental idea is that a domain expert who inspects an Armstrong table will easily spot violations of actually meaningful constraints that are currently perceived meaningless by the design team. The domain experts can point these violations out to the designers, who thus learn semantically meaningful constraints. These intuitions about the usefulness of Armstrong samples for the acquisition of semantically meaningful functional dependencies have been empirically validated in the literature (Langeveldt & Link 2010).

Example 2 *Suppose the design team for the application domain of Example 1 is certain that the FD Item,Size → Price is a semantically meaningful constraint. However, they are uncertain about other FDs, and keys in general. To consolidate their understanding, they create the Armstrong sample for* $\Sigma = \{$*Item,Size → Price*$\}$ *from Example 1 that they show to some domain experts. When inspecting the Armstrong bag they wonder why the price of* ¥70 *applies to both* average *and* small *sizes of* nigiri sake*.*

$$\frac{kX}{X \to Y}\ \text{(implication)} \qquad \frac{X \to Y \qquad kY}{kX}\ \text{(pullback)}$$

$$\frac{}{XY \to X}\ \text{(reflexivity)} \qquad \frac{X \to Y}{X \to XY}\ \text{(extension)} \qquad \frac{X \to Y \qquad Y \to Z}{X \to Z}\ \text{(transitivity)}$$

Table 2: A Minimal Finite Axiomatization $\mathfrak{F}$ of Keys and FDs over Bags

After some discussion, it turns out that the same items with the same price cannot have different sizes. As a consequence, the design team has newly gathered the semantically meaningful FD Item,Price → Size.

So far, however, the literature has mainly looked at Armstrong relations. In this paper, we will investigate structural and computational properties of Armstrong bags for the combined class of keys and functional dependencies. We will show how to transfer results known for functional dependencies over relations (Beeri et al. 1984, Mannila & Räihä 1986) to characterize and compute Armstrong bags for keys and functional dependencies. More precisely, using the notions of agree and maximal attribute sets we will characterize when a given bag is an Armstrong bag for a given set of keys and functional dependencies. Using this structural characterization, we will then establish an algorithm that computes an Armstrong bag for a given set of keys and functional dependencies. While the time-complexity of computing Armstrong bags is precisely exponential in the size of the given constraints in general, our algorithm computes an Armstrong bag whose number of rows is at most quadratic in the minimum number of rows required. Armstrong bags can be created from a given set of constraints. Vice versa, we may ask for a given bag for which set of constraints this bag is Armstrong. This is the problem of constraint discovery from data, and as it turns out, it is also very useful in the context of constraint acquisition.

Example 3 *Coming back to Example 2, the domain experts may have legacy data available, or may want to modify the data provided by the design team. For example, they provide the following bag to the design team:*

Item	*Size*	*Price*
takoyaki	*large*	*¥90*
takoyaki	*large*	*¥90*
nigiri sake	*large*	*¥90*
nigiri sake	*average*	*¥70*
nigiri toro	*small*	*¥70*
okonomiyaki	*small*	*¥300*

The designers would now welcome tools to discover the set of keys and functional dependencies satisfied by this bag. It turns out that a cover for this set consists of no keys and the two FDs Item,Size → Price and Item,Price → Size.

In our next contribution we show how to compute a cover for the set of keys and FDs that hold in a given bag. This allows the database designer in our running example, to discover the set of keys and FDs that hold in the bag of Example 3. The problem of dependency discovery is not only important in database design, but also in database re-organization and for query optimization, among others (Mannila & Räihä 1994). We can combine the schema-driven approach (computing Armstrong bags) with the sample-driven approach (discovering dependencies) to the discovery of keys and functional dependencies. That is, we compute small semantic samples of existing bags. A semantic sample of a given bag is a bag contained in the given bag that satisfies the same keys and functional dependencies. The size, however, of the semantic sample is usually much smaller than the size of the given bag. In the literature such semantic samples are also known as informative Armstrong samples (De Marchi & Petit 2007). The benefit of informative Armstrong samples is that they consist of real-world tuples. This is in contrast to Armstrong samples computed from a given constraint set only. In the latter, a design team would need to substitute artificial values by real-world domain values first, before presenting the sample to the domain experts. Yet, the combination of real-world domain values may still not contain a real-world tuple. Therefore, informative Armstrong samples take advantage of the additional input in form of legacy data.

1.3 Organization

We comment on related work in Section 2. The data model, including the framework of keys and FDs is defined in Section 3. We characterize the structure of Armstrong bags in Section 4, and also show how to compute Armstrong bags with a small number of tuples. The problem of dependency discovery for keys and FDs in bags is analyzed in Section 5. We conclude and comment on future work in Section 6.

2 Related Work

Data dependencies have been studied thoroughly in various data models, and for the purpose of this paper it is not useful to aim at a complete overview. Mainly, we will focus on work related to the results established in this paper, i.e., to the following areas concerning keys and FDs: *axiomatisations and implication problem*, *Armstrong samples*, *dependency discovery* and *informative Armstrong samples*.

Keys and FDs are concepts almost as old as the relational model of data itself (Codd 1970). Armstrong established the first axiomatization of FDs under set semantics (Armstrong 1974), now known as the *Armstrong axioms*. In fact, Armstrong showed that the Armstrong axioms are even strongly complete for the implication of FDs, i.e., for an arbitrary relation schema and an arbitrary set of FDs on that schema, he constructed a single finite set of tuples which satisfies precisely all implied FDs. That is the reason, why such specific relations became known as *Armstrong relations* and more generally as *Armstrong databases* for more general classes of data dependencies.

Armstrong databases constitute an invaluable tool for the validation of semantic knowledge, and a user-

friendly representation of integrity constraints. Armstrong relations have been deeply studied for keys (Demetrovics 1980, Thalheim 1989) and FDs (Armstrong 1974, Beeri et al. 1984, Demetrovics et al. 1998, Mannila & Räihä 1986). They have also been analyzed for various other classes of data dependencies. An excellent survey on Armstrong databases is (Fagin 1982*a*), and a brief review of recent results is (Link 2012). In (Hartmann et al. 2012) Armstrong databases for the combined class of keys and functional dependencies over partial bags were studied. Here, null values are permitted to occur in columns that are null-able. The class of keys and FDs over bags are a special case where no attribute is null-able. However, we establish a construction of Armstrong bags which generally requires only half the number of tuples as the Armstrong samples constructed in the special case of total bags in (Hartmann et al. 2012). That is, the special case of total bags enjoys an optimization in terms of the size of the Armstrong sample constructed, and we establish this optimization here. The concept of *informative Armstrong databases* was introduced (De Marchi & Petit 2007). These are small subsets of an existing database, satisfying exactly the same functional and inclusion dependencies.

Algorithms for *discovering FDs* from relations can be found in (Mannila & Räihä 1994). In particular, one of the algorithms uses hypergraph transversals. We will adapt the hypergraph transversal approach to the discovery of keys and FDs in bags. The characterizations of sets of keys and FDs satisfied by a relation in terms of hypergraph transversals was also observed in (Thi 1986). The problem of dependency discovery has also been studied for other data dependencies, e.g. multivalued dependencies, inclusion dependencies, excluded dependencies, embedded dependencies, branching and fractional dependencies, and FDs in XML. Data dependency discovery is an attractive problem in machine learning (Flach & Savnik 1999): the inference of general rules from instances of data. The problem is attractive since there always exists a set of dependencies that fits the instance exactly, and therefore dependency discovery can be solved exactly, while in inductive learning there is always a possibility of error (Mannila & Räihä 1994). Recently, the approximation variant of the dependency discovery problem has been studied (Giannella & Robertson 2004, Huhtala et al. 1999). In this setting, various measures for the error of a dependency in a relation are considered. These error measures have the value 0 if the dependency holds and a value close to 1 if the dependency clearly does not hold. To the best of our knowledge, dependency discovery has not been studied over bags yet.

3 Preliminaries

In this section we will define the underlying concepts of our study. These include the definition of schemata, relations and bags, but also keys and FDs. Subsequently, we will briefly summarize the notions of and some results on the implication and inferences for keys and FDs.

3.1 Schemata and Instances

A *bag schema* is a finite, non-empty set $\mathfrak{B}$ of attributes. Every attribute A of a bag schema $\mathfrak{B}$ is associated with a *domain* $dom(A)$, an at most countably infinite set of possible values. A *tuple over* $\mathfrak{B}$ is a function $t : \mathfrak{B} \to \cup_{A \in \mathfrak{B}} dom(A)$ such that for all $A \in \mathfrak{B}$ we have that $t(A) \in dom(A)$ holds. For a subset $X \subseteq \mathfrak{B}$ the *projection of a tuple* t over $\mathfrak{B}$ on X is the restriction $t(X)$ of t to X. For subsets X and Y of $\mathfrak{B}$ we write XY for the set union $X \cup Y$. If $X = \{A_1, \ldots, A_n\}$, then we may write $A_1 \cdots A_n$ for X. In particular, we may write simply A to denote the singleton $\{A\}$. A *bag over* $\mathfrak{B}$ is a finite multiset of tuples over $\mathfrak{B}$, usually denoted by $\mathfrak{b}$. We will use $\{\{\cdot\}\}$ to enumerate the elements of a bag explicitly, e.g. $\{\{t, t\}\}$ consists of two occurrences of the tuple t. A *relation over* $\mathfrak{B}$, usually denoted by r, is a finite set of tuples over $\mathfrak{B}$. In the context of a relation, we usually speak of a *relation schema*, usually denoted by R, rather than a bag schema. Commonly, we will speak of a *schema* S if we refer either to a relation schema or to a bag schema. Furthermore, we will speak of an *instance* if we refer either to a relation or to a bag. Instances can be illustrated as tables with each tuple of the instance corresponding to a row of the table. The attributes of the corresponding schema may be used as column headers.

Example 4 *Consider the bag schema* Order *with attributes* Item*,* Size *and* Price *from Example 1. We may choose the same domain STRING for all the attributes. Quite naturally, duplicate tuples occur in bags over* Order*: some customer's order might consist of 3 large takoyaki for the price of ¥90 each:*

Item	*Size*	*Price*
takoyaki	*large*	*¥90*
takoyaki	*large*	*¥90*
takoyaki	*large*	*¥90*

For example, the projection of the tuple (takoyaki, large, ¥90) on {Item,Price} is (takoyaki,¥90).

Integrity constraints are specified as semantic constraints on schemata. They enable us to model real-world instances of a schema by restricting the set of possible instances to those considered meaningful to the application at hand. One of the most fundamental classes of integrity constraints are keys and FDs. Keys enable us to uniquely identify tuples within an instance.

Definition 1 *Let* $\mathfrak{B}$ *be a bag schema. A* key *over* $\mathfrak{B}$ *is an expression* kX *where* X *is a non-empty subset of* $\mathfrak{B}$*. A bag* $\mathfrak{b}$ *over* $\mathfrak{B}$ *is said to* satisfy *the key* kX *over* $\mathfrak{B}$*, denoted by* $\models_{\mathfrak{b}} kX$*, if every pair of distinct tuples in* $\mathfrak{b}$ *deviates on some attribute in* X*, i.e., for all* $t_1, t_2 \in \mathfrak{b}$ *we have: if* $t_1 \neq t_2$*, then there is some* $A \in X$ *such that* $t_1(A) \neq t_2(A)$ *holds. In other words,* $\mathfrak{b}$ *satisfies the key* kX *if for every* $t_1, t_2 \in \mathfrak{b}$ *the following holds: if* $t_1(X) = t_2(X)$*, then* $t_1 = t_2$*.*

While any relation over the relation schema R satisfies the key kR, proper bags (those that are not a set) do not satisfy any key over the associated bag schema.

Example 5 *Consider the following bag* $\mathfrak{b}$ *over* Order*:*

Item	*Size*	*Price*
okonomiyaki	*average*	*¥450*
okonomiyaki	*average*	*¥450*

None of the subsets X *of* Order={Item,Size,Price} *satisfies the property that for all distinct* $t_1, t_2 \in \mathfrak{b}$ *we have* $t_1(X) \neq t_2(X)$*.*

In relational databases, i.e., when we consider relations over relation schemata, then the concept of a key can be generalized by the concept of a functional dependency.

Definition 2 *Let $\mathfrak{B}$ denote a bag schema. A* functional dependency *(FD) over $\mathfrak{B}$ is an expression $X \to Y$ where $X, Y \subseteq \mathfrak{B}$. A bag $\mathfrak{b}$ over $\mathfrak{B}$ is said to* satisfy *the FD $X \to Y$ over $\mathfrak{B}$, denoted by $\models_{\mathfrak{b}} X \to Y$, if for every pair of tuples in $\mathfrak{b}$ that match on all attributes in X also match on all attributes in Y, i.e., for all $t_1, t_2 \in \mathfrak{b}$ we have: if $t_1(X) = t_2(X)$, then $t_1(Y) = t_2(Y)$ holds.*

Example 6 *Consider again the bag schema* Order *with attributes* Item, Size *and* Price. *Intuitively, the same item of the same size should always have the same price. This "business rule" can be formalized as the functional dependency*

$$\text{Item, Size} \to \text{Price}.$$

Notice that the bags in Examples 4 and 5 satisfy this functional dependency. Since each instance is expected to satisfy this functional dependency, instances such as

Item	*Size*	*Price*
okonomiyaki	*average*	*¥450*
okonomiyaki	*average*	*¥500*

are prohibited from entering the database. Notice that it does not make sense to specify the functional dependency

$$\text{Item} \to \text{Price}$$

since there might be same items that have different prices (e.g. because they are different in size), for example

Item	*Size*	*Price*
okonomiyaki	*average*	*¥450*
okonomiyaki	*small*	*¥300*

The challenge for the database designer is to identify all keys and FDs that are meaningful to the application domain.

Example 7 *Note that the bags in Examples 4 and 5 satisfy every FD over* Order, *but violate every key over* Order. *No set of constraints that consists exclusively of FDs ever implied any key over bags.*

3.2 Implication and inference

For the design of a schema, constraints are commonly specified as semantic constraints on the intended instances of the schema. During the design process one needs to determine further constraints which are implied by the given ones. In the following, $\mathcal{C}$ denotes a class of integrity constraints, say the combined class of keys and FDs.

Definition 3 *Let S denote a schema, and let $\Sigma \cup \{\varphi\}$ be a set of integrity constraints of class $\mathcal{C}$ over S. We say that Σ* implies *φ, denoted by $\Sigma \models \varphi$, if and only if each instance over S that satisfies all $\sigma \in \Sigma$ also satisfies φ.*

In order to determine the logical consequences of a set of constraints one can utilize a syntactic approach by applying inference rules, e.g. those in Table 1. These *inference rules* have the form

$$\frac{\text{premise}}{\text{conclusion}},$$

and inference rules without any premises are called *axioms*.

Let $\Sigma \cup \{\varphi\}$ be a set of integrity constraints from a class $\mathcal{C}$, all defined over a schema S. Furthermore, we use $\mathfrak{S}$ to denote a set of inference rules. A finite sequence $\gamma = [\varphi_1, \ldots, \varphi_n]$ of integrity constraints from $\mathcal{C}$ is called an *inference from Σ by $\mathfrak{S}$* if and only if each φ_i is either an element of Σ or is obtained by applying one of the rules of $\mathfrak{S}$ to appropriate elements of $\{\varphi_1, \ldots, \varphi_{i-1}\}$. We say that the inference γ infers φ_n, i.e. the last element of the sequence γ, and write $\Sigma \vdash_{\mathfrak{S}} \varphi_n$. Let $\Sigma^+_{\mathfrak{S}} = \{\varphi \mid \Sigma \vdash_{\mathfrak{S}} \varphi\}$ denote the *syntactic closure* of Σ under inferences by $\mathfrak{S}$. An inference rule is called *sound* if the set of constraints in the premise of the rule implies the dependency in the conclusion. The set $\mathfrak{S}$ is called *sound* for the implication of constraints in $\mathcal{C}$ if for every schema S and for every set Σ of constraints in $\mathcal{C}$ over S we have $\Sigma^+_{\mathfrak{S}} \subseteq \Sigma^* = \{\varphi \mid \Sigma \text{ implies } \varphi\}$. The set $\mathfrak{S}$ is called *complete* for the implication of constraints in $\mathcal{C}$ if for every schema S and for every set Σ of constraints in $\mathcal{C}$ over S we have $\Sigma^* \subseteq \Sigma^+_{\mathfrak{S}}$. The (finite) set $\mathfrak{S}$ is called a (finite) *axiomatization* for the implication of constraints in $\mathcal{C}$ if it is both sound and complete for the implication of constraints in $\mathcal{C}$. The rules of Table 1 form finite axiomatizations for the implication of keys, and of FDs, respectively, over relations (Armstrong 1974). The rules of Table 2 form a finite axiomatization for the implication of keys and FDs over bags (Köhler & Link 2010).

3.3 Interactions in relations and bags

Let us fix a relation schema R. A key kX over R implies the functional dependency $X \to R$ since in every relation over R that satisfies the key kX there can never be two distinct tuples that have matching values on all attributes in X. Vice versa, the functional dependency $X \to R$ also implies the key kX over R: in every relation over R no two distinct tuples can have matching values on all attributes in R, i.e., they must have non-matching values on some attribute in X by means of the functional dependency $X \to R$. Hence, a key kX is equivalent to the functional dependency $X \to R$ over the relation schema R, in the sense that they are satisfied by the same relations r over R. Consequently, FDs subsume the concept of keys in the context of relational databases.

Let us now fix a bag schema $\mathfrak{B}$. Keys kX over $\mathfrak{B}$ still imply FDs $X \to \mathfrak{B}$ over $\mathfrak{B}$. However, the reverse direction does not hold: take the bag $\mathfrak{b} = \{\{t, t\}\}$ where t denotes some tuple over $\mathfrak{B}$. The same bag $\mathfrak{b}$ shows that there does not necessarily need to be any key over $\mathfrak{B}$ that is satisfied by a bag. This situation is exemplified by Example 6. The functional dependency *Item,Size* $\to$ *Price* does not imply the key k\{*Item,Size*\} on the bag schema {*Item, Size, Price*}.

4 Armstrong Bags

In this section we will investigate structural and computational properties of Armstrong bags for sets of keys and functional dependencies. Starting from the definition of an Armstrong bag we show first that any set of keys and functional dependencies has an Armstrong bag. Using the essential notions of agree sets and maximal attribute sets, we then establish sufficient and necessary conditions for a given bag to be Armstrong for a given set of keys and FDs. Based on these conditions we then establish an algorithm that computes an Armstrong bag for any given set of keys and functional dependencies. The number of tuples in the computed Armstrong bag is guaranteed

to be at most quadratic in the minimum number of tuples required by an Armstrong bag. The results extend the structural and computational properties of Armstrong relations known for sets of functional dependencies over relations (Beeri et al. 1984, Mannila & Räihä 1986). They also simplify the structural and computational properties of Armstrong samples known for sets of keys and functional dependencies over partial bags (Hartmann et al. 2012). In particular, the construction described in this paper required about half the number of tuples as the construction from (Hartmann et al. 2012) for partial bags applied to the special case of (total) bags.

4.1 Existence of Armstrong Bags

Armstrong samples are concise "user-friendly" representations of sets of data dependencies, they have important applications in the acquisition and validation of these dependencies (Beeri et al. 1984, De Marchi & Petit 2007, Fagin 1982*a*, Link 2012, Mannila & Räihä 1986). We begin with the definition of Armstrong bags.

Definition 4 *Let $\mathcal{C}$ denote a class of constraints. A bag $\mathfrak{b}$ over bag schema $\mathfrak{B}$ is said to be a* $\mathcal{C}$-Armstrong bag *for a given set Σ of constraints in $\mathcal{C}$ over $\mathfrak{B}$ if and only if for every constraint $\varphi \in \mathcal{C}$ over $\mathfrak{B}$ the following holds: $\mathfrak{b}$ satisfies φ if and only if Σ implies φ. The class $\mathcal{C}$ is said to* enjoy *Armstrong bags if and only if for every bag schema $\mathfrak{B}$ and every set Σ of constraints in $\mathcal{C}$ over $\mathfrak{B}$ there is a bag $\mathfrak{b}$ over $\mathfrak{B}$ that is $\mathcal{C}$-Armstrong for Σ.*

The finite axiomatization $\mathfrak{F}$ from Table 2 for the implication of keys and FDs over bags was established in (Köhler & Link 2010). We use the construction in the completeness proof to establish the fact that keys and FDs enjoy Armstrong bags. Note that such a result should not be taken for granted. The class of functional and inclusion dependencies over relations, for example, does not enjoy Armstrong databases; and neither does the class of general keys and functional dependencies over partial bags (Hartmann et al. 2012). We assume that the domain of every attribute of any given bag schema has sufficiently many domain values.

Let $\mathfrak{B}$ be a bag schema, and Σ a set of keys and FDs defined on $\mathfrak{B}$. For every key and every functional dependency φ such that $\varphi \notin \Sigma^+_{\mathfrak{F}}$, we construct a bag $\mathfrak{b}_{\Sigma,\varphi}$ precisely as in the proof of Theorem 7 in (Köhler & Link 2010).

Let $\mathfrak{b}$ denote the union of the $\mathfrak{b}_{\Sigma,\varphi}$ for all $\varphi \notin \Sigma^+_{\mathfrak{F}}$. However, for every attribute $A \in \mathfrak{B}$ and for every element $d \in dom(A)$ we require that d occurs in a two-element bag $\mathfrak{b}_{\Sigma,\varphi}$ for at most one φ. Note that this construction is possible since the domain of every attribute is assumed to have sufficiently many elements. However, for each attribute $A \in \emptyset^+_{\mathfrak{F}}$ (that is, for each attribute A such that $\emptyset \to A \in \Sigma^+_{\mathfrak{F}}$), we require that for all $t, t' \in \mathfrak{b}$ we have $t(A) = t'(A)$, i.e., every A entry in every tuple in $\mathfrak{b}$ is the same value. It follows that $\mathfrak{b}$ satisfies the keys and FDs in Σ, and violates every key and every FD over $\mathfrak{B}$ not implied by Σ. Therefore, we obtain the following result.

Theorem 1 *Keys and FDs enjoy Armstrong bags.*

4.2 Structural Characterization

Armstrong bags for a given set Σ of keys and FDs must violate every FD $XY \to A$ not implied by Σ. Whenever $XY \to A$ is violated, then so is $X \to A$. Therefore, it suffices to violate the FDs $X \to A$ where X is maximal with the property that $X \to A$ is not implied by Σ. For an attribute $A \in \mathfrak{B}$ let therefore

$$max(A) = \{X \mid X \subseteq \mathfrak{B}, X \to A \notin \Sigma^*, \forall Y \subseteq \mathfrak{B}(X \subset Y \Rightarrow (Y \to A \in \Sigma^*))\}$$

denote the set of all maximal attribute subsets of $\mathfrak{B}$ on which A is not functionally dependent (Mannila & Räihä 1986). Furthermore, let $MAX(\mathfrak{B}) = \bigcup_{A \in \mathfrak{B}} max(A)$ and $CL(\mathfrak{B}) = \{X \mid X \subseteq \mathfrak{B}, X^*_\Sigma = X\}$ (Mannila & Räihä 1986). Here, $X^*_\Sigma = \{A \in \mathfrak{B} \mid X \to A \in \Sigma^*\}$ denotes the attribute closure of X with respect to Σ. It follows that $MAX(\mathfrak{B}) \subseteq CL(\mathfrak{B})$, and $\mathfrak{B} \in CL(\mathfrak{B})$. It is known that $MAX(\mathfrak{B}) = GEN(\mathfrak{B})$ (Mannila & Räihä 1986) where $GEN(\mathfrak{B})$ denotes the unique minimal subfamily of generators in $CL(\mathfrak{B})$ such that each element of $CL(\mathfrak{B})$ can be expressed as an intersection of sets in $GEN(\mathfrak{B})$ ($\mathfrak{B}$ is the intersection of an empty collection of sets).

For some bag $\mathfrak{b}$ over $\mathfrak{B}$ let $t, t' \in \mathfrak{b}$. The *agree set* of t, t' (Mannila & Räihä 1986) is defined as $ag(t, t') = \{A \mid A \in \mathfrak{B}, t(A) = t'(A)\}$. Moreover, the *agree set of* $\mathfrak{b}$ (Mannila & Räihä 1986) is defined as

$$ag(\mathfrak{b}) = \{ag(t, t') \mid t, t' \in \mathfrak{b}, t \neq t'\}.$$

For relations the following result is known (Beeri et al. 1984, Theorem 6.1). Let R denote a relation schema, and Σ a set of FDs over R. Then for every relation r over R we have: r is an Armstrong relation with respect to Σ if and only if $MAX(R) \subseteq ag(r) \subseteq CL(R)$.

Note that for every bag schema $\mathfrak{B}$ and every bag $\mathfrak{b}$ over $\mathfrak{B}$ we have $\mathfrak{B} \in ag(\mathfrak{b})$ precisely if $\mathfrak{b}$ is not a relation. This indicates that the attribute set $\mathfrak{B}$ itself is required to characterize Armstrong bags, as illustrated by the following example.

Example 8 *Consider the bag schema* Order *and the set Σ that contains the functional dependency Item,Size $\to$ Prize only. The relation*

Item	*Size*	*Price*
takoyaki	*large*	*¥90*
nigiri sake	*large*	*¥90*
nigiri sake	*average*	*¥70*
nigiri sake	*small*	*¥70*
okonomiyaki	*small*	*¥300*

satisfies the condition $MAX(\mathfrak{B}) \subseteq ag(b) \subseteq CL(\mathfrak{B})$, but it is not an Armstrong bag with respect to Σ. For instance, it satisfies the key k{Item,Size} which is not implied by Σ.

An Armstrong bag for a set Σ of keys and functional dependencies must also violate all keys not implied by Σ. Suppose kX is not implied by Σ over bag schema $\mathfrak{B}$. If $X \to \mathfrak{B}$ is not implied by Σ, then $X \to A$ is not implied by Σ for some $A \in \mathfrak{B} - X$. Hence, $X \subseteq M \in max(A)$. In this case, any bag that violates the FD $M \to A$ will also violate the key kX, including any Armstrong bag for Σ. Consider now the case where $X \to \mathfrak{B}$ is implied by Σ. Since kX is not implied by Σ it follows by the soundness of the pullback rule that the weakest possible key $k\mathfrak{B}$ is also not implied by Σ. This, however, means that Σ does not contain any key (otherwise $k\mathfrak{B}$ would be implied by Σ).

We will now extend the structural characterization of Armstrong relations for FDs (Beeri et al. 1984,

Mannila & Räihä 1986) to a structural characterization of Armstrong bags for keys and functional dependencies.

Theorem 2 *Let $\mathfrak{b}$ be a bag over $\mathfrak{B}$, and let $\Sigma = \Sigma_k \cup \Sigma_f$ be a set of keys and FDs over $\mathfrak{B}$. Then $\mathfrak{b}$ is an Armstrong bag for Σ if and only if both of the following conditions are satisfied: i) $MAX(\mathfrak{B}) \subseteq ag(\mathfrak{b}) \subseteq CL(\mathfrak{B})$, and ii) $\mathfrak{B} \in ag(\mathfrak{b})$ if and only if $\Sigma_k = \emptyset$.*

We now give the proof of Theorem 2. For that purpose we distinguish between two different cases. Let $\Sigma_k \neq \emptyset$. If $\mathfrak{b}$ is an Armstrong bag for Σ, then it is a relation as it satisfies some key. Due to the characterization of Armstrong relations established in (Beeri et al. 1984, Theorem 6.1), condition i) is satisfied. Condition ii) is also satisfied since a relation cannot have duplicate tuples. If $\mathfrak{b}$ satisfies conditions i) and ii), then it is a relation. Due to the characterization of Armstrong relations established in (Beeri et al. 1984, Theorem 6.1), $\mathfrak{b}$ must be an Armstrong relation for Σ.

It remains to consider the case where $\Sigma_k = \emptyset$. Assume first that $\mathfrak{b}$ satisfies conditions i) and ii). Due to condition ii), $\mathfrak{b}$ violates every possible key over $\mathfrak{B}$. Condition i) ensures that $\mathfrak{b}$ satisfies precisely those FDs implied by Σ_f, cf. (Beeri et al. 1984, Theorem 6.1). Assume now that $\mathfrak{b}$ is an Armstrong bag for Σ. Since $\Sigma_k = \emptyset$ it follows that $\mathfrak{b}$ violates every possible key over $\mathfrak{B}$, in particular $\mathfrak{B}$ itself. Consequently, condition ii) is satisfied. Condition i) follows from the fact that $\mathfrak{b}$ satisfies precisely those FDs implied by Σ_f, cf. (Beeri et al. 1984, Theorem 6.1).

4.3 Computational Properties

Given a first approximation of the target data dependencies in form of a set Σ of explicitly specified keys and FDs on a bag schema $\mathfrak{B}$, a design team may want to validate this approximation with domain experts or users of the target database. These domain experts and users usually do not have the background to understand the meaning of Σ. Instead, the design team may let a computer generate an Armstrong bag with respect to Σ, substitute the artificially generated values by real domain values, and then ask domain experts and users to validate the bag. Example 3 showed how visitors and owners of the restaurant modified the first approximation of the target database by the design team. This exemplifies how the semantics of the underlying application domain can easily be conveyed, and tested by Armstrong bags.

We would like to have an algorithm that generates an Armstrong bag with a relatively small number of tuples since smaller examples are usually easier to comprehend. The following algorithm is an extension of the one presented for the case of relations (Mannila & Räihä 1986).

The Armstrong bag of Example 1 with respect to the functional dependency *Item, Size* $\rightarrow$ *Price* is generated by Algorithm 1 when the sets in $MAX(\mathfrak{B})$ are processed in order: {*Size,Price*}, {*Item*}, {*Item,Price*} and {*Size*}.

The correctness of Algorithm 1 follows from Theorem 2.

Theorem 3 *Algorithm 1 returns an Armstrong bag $\mathfrak{b}$ with respect to Σ, and $\mathfrak{b}$ contains $|MAX(\mathfrak{B})| + 1$ tuples if $\Sigma_k \neq \emptyset$, and $|MAX(\mathfrak{B})|+2$ tuples otherwise.*

Algorithm 1 Armstrong Bag Computation

```
1:  procedure ARMSTRONG-BAG(𝔅, Σ_k, Σ_FD)
2:      compute MAX(𝔅);
3:      for all A ∈ T do
4:          t_0(A) ← c_{A,0};
5:      end for
6:      if Σ_k = ∅ then
7:          𝔟 ← {{t_0, t_0}};
8:      else
9:          𝔟 ← {t_0};
10:     end if
11:     i ← 1;
12:     for all W ∈ MAX(𝔅) do
13:         for all A ∈ 𝔅 do
14:             if A ∈ W then
15:                 t_i(A) ← t_{i−1}(A);
16:             else
17:                 t_i(A) ← c_{A,i};
18:             end if
19:         end for
20:         𝔟 ← 𝔟 ∪ {t_i};
21:         i ← i + 1;
22:     end for
23:     return 𝔟;
24: end procedure
```

If $\Sigma_k \neq \emptyset$, then we are in the case of relations. There, line 8 ensures that no duplicate tuples will occur in the initialization of $\mathfrak{b}$. The remaining steps are the exact steps of the algorithm in (Mannila & Räihä 1986).

If $\Sigma_k = \emptyset$, then Theorem 2 tells us that $\mathfrak{b}$ must contain distinct tuples with agree set $\mathfrak{B}$. Therefore, line 6 ensures that such a duplicate tuple will occur in the initialization of $\mathfrak{b}$. The remaining steps ensure that Algorithm 1 produces an Armstrong bag that satisfies precisely all the FDs implied by Σ_f (Mannila & Räihä 1986).

The time complexity of finding an Armstrong bag, given a set of keys and FDs, is precisely exponential in the size of the given constraints. That is, (1) there is an algorithm for obtaining an Armstrong bag, given the set Σ of keys and FDs, where the running time of the algorithm is exponential in the size of the constraints; and (2) there is a set Σ of keys and FDs in which the number of tuples in each Armstrong bag for Σ is exponential. The result is a consequence of (Beeri et al. 1984, Theorem 7.1).

Theorem 4 *The complexity of finding an Armstrong bag with respect to a given set of keys and FDs over a bag schema $\mathfrak{B}$ is precisely exponential in the size of the given keys and FDs.*

Statement (2) above follows immediately from the construction in (Beeri et al. 1984, Theorem 7.1) since every relation is a bag. Statement (1) above follows also immediately from the construction in (Beeri et al. 1984, Theorem 7.1) in the case where $\Sigma_k \neq \emptyset$. For the remaining case where $\Sigma_k = \emptyset$ we simply pick a tuple t from the Armstrong relation r constructed in (Beeri et al. 1984, Theorem 7.1) and add another tuple t' to r which coincides with t on all attributes. It follows that $\mathfrak{b} := r \cup \{\{t'\}\}$ is an Armstrong bag for Σ.

Note that this is a worst-case analysis. There are also sets of keys and FDs where the number of tuples in a minimum-sized Armstrong bag is logarithmic in the size of an optimal cover of the constraint set. Consequently, there is no best representation of constraints, i.e., either in form of a constraint set or in form of an Armstrong sample. Indeed, it is best

to have both representations. Armstrong bags reveal semantically meaningful constraints perceived semantically meaningless, and representations in form of constraint sets reveal semantically meaningless constraints perceived semantically meaningful.

5 Dependency discovery

In Example 3 the guests and owners of our restaurant example have produced another bag as their "model" of data that they think is typical for the application domain at hand. The design team is faced with the problem of extracting those keys and FDs that hold in this bag. Indeed, the problem the design team faces is known as the problem of dependency discovery. In our case, the team would like an algorithm that computes a cover of the set of keys and FDs that hold in the given bag. This is the reverse problem to computing an Armstrong bag given a set of keys and FDs. In this section, we will show how an algorithm for functional dependency discovery under set semantics can be adapted to key and functional dependency discovery under bag semantics. This algorithm will make extensive use of hypergraph transversals (Eiter & Gottlob 1995, Mannila & Räihä 1994). We argue that even in the presence of relations the discovery of FDs should include a discovery of all minimal keys, followed by the discovery of the remaining FDs not implied by the minimal keys.

5.1 Hardness of Dependency Discovery

We start this section by introducing some notation. Let $\mathfrak{b}$ denote a bag over bag schema $\mathfrak{B}$. The set of all keys and FDs holding in $\mathfrak{b}$ is denoted by $dep(\mathfrak{b}) = dep_k(\mathfrak{b}) \cup dep_f(\mathfrak{b})$, i.e., $X \in dep_k(\mathfrak{b})$ if and only if $\models_{\mathfrak{b}} kX$, and $X \to Y \in dep_f(\mathfrak{b})$ if and only if $\models_{\mathfrak{b}} X \to Y$ over $\mathfrak{B}$. If Σ and Σ' are equivalent sets of keys and FDs, i.e., all the dependencies of Σ imply those of Σ' and vice versa, then we say that Σ is a *cover* of Σ' (and Σ' is a *cover* of Σ). In general, $dep(\mathfrak{b})$ has several covers of varying size. We now adapt some of the results on the complexity of dependency discovery to bags. The first result shows that for some bags over n attributes all the covers of their dependency sets are of exponential size in n.

Theorem 5 *For each n there is a bag $\mathfrak{b}$ over $\mathfrak{B}$ such that $|\mathfrak{B}| = n, |\mathfrak{b}| = \mathcal{O}(n)$, and each cover of $dep(\mathfrak{b})$ has $\Omega(2^{n/2})$ dependencies.*

Since every relation is a bag the corollary follows from the same statement that was established previously for relations (Mannila & Räihä 1992).

In practice, bags that have inherently large covers should be rare. Corollary 5 shows that the results of dependency discovery can be large. One can also show that it is hard to identify these dependency sets.

Theorem 6 *The problem of determining whether a bag has a key containing at most k attributes is* NP*-complete.*

The problem is clearly in *NP*. The corresponding problem for relations has been shown to be *NP*-hard by a reduction from the *Vertex Cover Problem* which is *NP*-complete (Beeri et al. 1984). The same reduction applies here as every relation is a bag.

Note that the dependency discovery problem for keys in relations (and therefore bags) is also inherently exponential in the number of attributes. Even though the dependency discovery problems are inherently exponential, Mannila and Räihä developed a useful and practical algorithm for inferring FDs from relations. This algorithm has demonstrated a satisfactory efficiency when being applied to "real-life" databases (Mannila & Räihä 1994). We will now discuss the use of this algorithm in bags, and also discuss how we can discover useful covers directly.

5.2 Minimal Left-hand Sides & Transversals

A simple reduction of the key and functional dependency discovery problem for bags to the functional dependency discovery problem in sets works as follows: (1) we scan the input bag $\mathfrak{b}$ for duplicate tuples, (2) if such duplicate tuples exist, then $\emptyset$ is a cover of $dep_k(\mathfrak{b})$, else $\{\mathfrak{B}\}$ is a cover of $dep_k(\mathfrak{b})$, and (3) we compute a cover of $dep_f(\mathfrak{b})$ by applying any algorithm for functional dependency discovery to the input $\mathfrak{b}$, e.g. those in (Mannila & Räihä 1994).

However, the set $\textit{min-key}(\mathfrak{b}) = \{X \in \mathfrak{B} \mid \models_{\mathfrak{b}} X \wedge \neg\exists Y \subset X(\models_{\mathfrak{b}} Y)\}$ of all minimal keys that hold in $\mathfrak{b}$ is a more desirable cover of $dep_k(\mathfrak{b})$ that has considerable advantages for database design, maintenance, query optimization and database tuning. The computation of $\textit{min-key}(\mathfrak{b})$ would also not increase the exponential time complexity of computing a cover of $dep_f(\mathfrak{b})$. Hence, we aim to find a cover of $dep(\mathfrak{b})$ that contains $\textit{min-key}(\mathfrak{b})$, and where the set of remaining FDs (those that hold in $\mathfrak{b}$ but are not implied by $\textit{min-key}(\mathfrak{b})$) is small. Alternatively to this strategy, one may compute all minimal keys from the cover obtained by the simple reduction above. However, unless *P=NP*, it takes exponential time to compute a minimal key, given a set of FDs. Thus, we prefer to discover all minimal keys directly from the input bag.

To keep things consistent with previous work (Mannila & Räihä 1994) we only consider *standard* FDs in this section, i.e., FDs $X \to A$ where $X \neq \emptyset$ and $A \in \mathfrak{B}$. Given $\mathfrak{B}$, and given an attribute $A \in \mathfrak{B}$, denote by $lhs(A)$ (left-hand sides of A) the set of minimal nontrivial attribute sets $X \subseteq \mathfrak{B}$ such that $X \to A$ is implied by Σ. That is, $lhs(A) = \{X \subseteq \mathfrak{B} \mid \Sigma \models X \to A \wedge \forall Y \subset X(\Sigma \not\models Y \to A)\}$.

We borrow some material from hypergraphs (Eiter & Gottlob 1995). A collection $\mathfrak{H}$ of subsets of $\mathfrak{B}$ is a *(simple) hypergraph*, if no element of $\mathfrak{H}$ is empty and if $X, Y \in \mathfrak{H}$ and $X \subseteq Y$ imply $X = Y$. The elements of $\mathfrak{H}$ are called *edges* of the hypergraph, and the elements of $\mathfrak{B}$ are called the *vertices* of the hypergraph. We first note that $lhs(A)$ is a hypergraph.

Given a simple hypergraph $\mathfrak{H}$ on $\mathfrak{B}$, a *transversal* T of $\mathfrak{H}$ is a subset of $\mathfrak{B}$ intersecting all the edges of $\mathfrak{H}$, that is, $T \cap E \neq \emptyset$ for all $E \in \mathfrak{H}$. Transversals are also called *hitting sets*. A *minimal transversal* of $\mathfrak{H}$ is a transversal T such that no $T' \subset T$ is a transversal. The collection of minimal transversals of $\mathfrak{H}$ is denoted by $Tr(\mathfrak{H})$. It is a hypergraph on $\mathfrak{B}$. The next algorithm computes $Tr(\mathfrak{H})$ for a simple hypergraph $\mathfrak{H}$ (Mannila & Räihä 1994).

Algorithm 2 Minimal Transversals of Hypergraphs

```
1: procedure TRANSVERSAL(𝔥 = (V, E))
2:     Tr(𝔥) ← {∅};
3:     for all E ∈ 𝔥 do
4:         Tr(𝔥) ← {X ∪ {e} | X ∈ Tr(𝔥) ∧ e ∈ E};
5:         Tr(𝔥) ← {X ∈ Tr(𝔥) |
6:                         ¬∃Y ∈ Tr(𝔥)(Y ⊂ X)};
7:     end for
8:     return Tr(𝔥);
9: end procedure
```

5.3 Dependency Discovery Algorithm

Let $disag'(\mathfrak{b}) = \{\mathfrak{B} - ag(t,t') \mid t,t' \in \mathfrak{b} \wedge t \neq t'\}$ denote the complements of agree sets of distinct tuples. Furthermore, let $disag(\mathfrak{b}) = \{X \in disag'(\mathfrak{b}) \mid \neg\exists Y \in disag'(\mathfrak{b})(Y \subset X)\}$ contain the minimal sets of $disag'(\mathfrak{b})$. We then have that $min\text{-}key(\mathfrak{b}) = Tr(disag(\mathfrak{b}))$ (Thi 1986).

Let $disag'(\mathfrak{b}, A) = \{\mathfrak{B} - ag(t,t') \mid t,t' \in \mathfrak{b} \wedge t[A] \neq t'[A]\}$ denote the complements of agree sets of those tuples that do not match on the attribute A. Note that $disag'(\mathfrak{b}, A) = \{W \in disag'(\mathfrak{b}) \mid A \in W\}$. Furthermore, let $disag(\mathfrak{b}, A) = \{X \in disag'(\mathfrak{b}, A) \mid \neg\exists Y \in disag'(\mathfrak{b}, A)(Y \subset X)\}$ contain the minimal sets of $disag'(\mathfrak{b}, A)$. Then it is known that $lhs(A) = Tr(disag(\mathfrak{b}, A))$ (Mannila & Räihä 1994, Theorem 5).

Next we present the key and functional dependency discovery algorithm. Lines 2-8 compute the minimal keys that hold in the input bag $\mathfrak{b}$, lines 9-14 produce a cover of the FDs that hold in $\mathfrak{b}$, and lines 15-19 compute a non-redundant cover of the set of FDs in the presence of the minimal keys. Other covers might be computed instead (Maier 1980).

Algorithm 3 Dependency Discovery

```
1:  procedure DISCOVERY(𝔅, 𝔟)
2:      J_𝔅 := {𝔅 − ag(t,t') | t,t' ∈ 𝔟 ∧ t ≠ t'};
3:      if ∅ ∈ J_𝔅 then
4:          Σ_k = ∅;
5:      else
6:          K_𝔅 := {W ∈ J_𝔅 | ¬∃V ∈ J_𝔅(V ⊂ W)};
7:          Σ_k = Tr(K_𝔅);
8:      end if
9:      for all A ∈ 𝔅 do
10:         J_A := {W ∈ J_𝔅 | A ∈ W};
11:         K_A := {W ∈ J_A | ¬∃V ∈ J_A(V ⊂ W)};
12:         L_A := Tr(K_A);
13:     end for
14:     Σ_f := {X → A | A ∈ 𝔅 ∧ X ∈ L_A};
15:     for all σ ∈ Σ_f do
16:         if Σ_k ∪ Σ_f − {σ} ⊨ σ then
17:             Σ_f ← Σ_f − {σ};
18:         end if
19:     end for
20:     return Σ_k ∪ Σ_f;
21: end procedure
```

Example 9 *Let $\mathfrak{b}$ denote the bag from Example 3 that is input to Algorithm 3. The set $J_{\mathfrak{B}}$ contains the sets $\emptyset$,* {Item}, {Item, Size, Price}, {Size, Price}, {Item, Size}, *and* {Item, Price}. *Consequently, line 7 returns $\Sigma_k = \emptyset$. Line 11 generates the following K-families:*

- $K_{Item} = \{Item\}$,
- $K_{Size} = \{Size, Price; Item, Size\}$,
- $K_{Price} = \{Size, Price; Item, Price\}$.

The L-families from line 12 are:

- $L_{Item} = \{Item\}$,
- $L_{Size} = \{Size; Item, Price\}$,
- $L_{Price} = \{Price; Item, Size\}$.

Line 14 produces the FDs Item → Item; Size → Size; Item, Price → Size; Price → Price; and Item, Size → Price. Finally, lines 15-19 produce $\Sigma_f = \{$Item, Price → Size; Item, Size → Price$\}$.

Algorithm 3 compute directly all minimal keys that hold in $\mathfrak{b}$. That is, there is no need to apply an exponential time algorithm to compute the set of minimal keys from a given set of FDs subsequently after the dependency discovery. We exemplify this by using an example from (Mannila & Räihä 1994).

Example 10 *Let r denote the following relation*

E(mployee)	*D(epartment)*	*M(anager)*	*S(alary)*
Smith	*Toys*	*Jones*	*200*
Wilson	*Admin*	*Brown*	*300*
Barnes	*Toys*	*Jones*	*300*

.

Algorithm 3 produces the following output on input r: $\Sigma_k = \{E, DS, MS\}$ and $\Sigma_f = \{D \to M, M \to D\}$. On the other hand, the Algorithm from (Mannila & Räihä 1994) would produce the following cover: $\{E \to D, E \to M, E \to S, D \to M, M \to D, DS \to E\}$.

Example 10 illustrates another application area of dependency discovery. While query optimization can benefit from the data dependencies enforced on the schema, dependency discovery can also infer additional dependencies that hold accidentally in the current instance. This provides further useful information to optimize queries.

Theorem 7 *Algorithm 3 computes a cover for $dep(\mathfrak{b})$ in time polynomial in $|\mathfrak{b}|$, $|\mathfrak{B}|$, $\prod_{W \in disag(\mathfrak{b})} |W|$ and the $\prod_{W \in disag(\mathfrak{b}, A)} |W|$.*

The size of the J_A-family is bounded by that of $J_{\mathfrak{B}}$. The size of $J_{\mathfrak{B}}$ is $\mathcal{O}(|\mathfrak{b}|^2)$. The K_A-family is a subcollection of the J_A-family, and can be found in time polynomial in $|\mathfrak{b}|$ and $|\mathfrak{B}|$. Computing the sets in $Tr(K_{\mathfrak{B}})$ and the L_A can be done in time polynomial with respect to $\prod_{W \in disag(\mathfrak{b})} |W|$ and $\prod_{W \in disag(\mathfrak{b}, \mathfrak{A})} |W|$, respectively, by using Algorithm 2.

6 Conclusion and Future Work

Duplicate detection has evolved into its own field of research (Naumann & Herschel 2010). In bags dependencies interact differently than they do in relations. As duplicates occur in real database systems, it is desirable to extend existing results from relations to bags. We have established a schema-driven and a data-driven approach to the discovery of keys and FDs in bags. For the schema-driven approach we compute Armstrong bags from a given set of keys and FDs. Armstrong bags visualize and communicate current perceptions of the domain semantics between stakeholders of the target database. This can lead to the discovery of semantically meaningful constraints previously perceived meaningless. For the data-driven approach we mine given bags for the set of keys and FDs it satisfies. These algorithms can be combined to generate informative Armstrong bags from a given bag, and serve as small semantic data samples to visualize the semantics currently present in a database.

For future work it would be interesting to develop database normalization algorithms that apply to bags, e.g., for 3NF, BCNF, or 4NF decompositions (Arenas & Libkin 2005, Biskup et al. 1979, Ferrarotti et al. 2012, Kolahi 2007, Vincent 1999). It is not clear what a lossless decomposition of a bag constitutes. The performance of Algorithm 3 can be improved (Mannila & Räihä 1994), and different types

of FD discovery algorithms should be adapted to the setting of bags. See (Liu et al. 2012) for a recent survey on dependency discovery algorithms.

References

Arenas, M. & Libkin, L. (2005), 'An information-theoretic approach to normal forms for relational and XML data', *J. ACM* **52**(2), 246–283.

Armstrong, W. W. (1974), 'Dependency structures of database relationships', *Information Processing* **74**, 580–583.

Beeri, C., Dowd, M., Fagin, R. & Statman, R. (1984), 'On the structure of Armstrong relations for functional dependencies', *J. ACM* **31**(1), 30–46.

Biskup, J., Dayal, U. & Bernstein, P. (1979), Synthesizing independent database schemas, *in* 'Proceedings of SIGMOD', pp. 143–151.

Biskup, J. & Lochner, J. (2008), 'Reducing inference control to access control for normalized database schemas', *Inf. Proc. Letters* **106**(1), 8–12.

Codd, E. F. (1970), 'A relational model of data for large shared data banks', *Commun. ACM* **13**(6), 377–387.

De Marchi, F. & Petit, J.-M. (2007), 'Semantic sampling of existing databases through informative Armstrong databases', *Inf. Syst.* **32**(3), 446–457.

Demetrovics, J. (1980), 'On the equivalence of candidate keys with Sperner systems', *Acta Cybern.* **4**, 247–252.

Demetrovics, J., Katona, G. O. H., Miklós, D., Seleznjev, O. & Thalheim, B. (1998), 'Asymptotic properties of keys and functional dependencies in random databases', *Theor. Comput. Sci.* **190**(2), 151–166.

Diederich, J. & Milton, J. (1988), 'New methods and fast algorithms for database normalization', *ACM Trans. Database Syst.* **13**(3), 339–365.

Eiter, T. & Gottlob, G. (1995), 'Identifying the minimal transversals of a hypergraph and related problems', *SIAM J. Comput.* **24**(6), 1278–1304.

Fagin, R. (1982*a*), Armstrong databases, Technical Report RJ3440(40926), IBM Research Laboratory, San Jose, California, USA.

Fagin, R. (1982*b*), 'Horn clauses and database dependencies', *J. ACM* **29**(4), 952–985.

Ferrarotti, F., Hartmann, S., Köhler, H., Link, S. & Vincent, M. W. (2012), Foundations for a Fourth normal form over SQL-like databases, *in* 'Conceptual Modelling and Its Theoretical Foundations', Vol. 7260 of *Lecture Notes in Computer Science*, Springer, pp. 85–100.

Flach, P. & Savnik, I. (1999), 'Database dependency discovery: A machine learning approach', *AI Commun.* **12**(3), 139–160.

Giannella, C. & Robertson, E. (2004), 'On approximation measures for functional dependencies', *Inf. Syst.* **29**(6), 483–507.

Hartmann, S., Kirchberg, M. & Link, S. (2012), 'Design by example for SQL table definitions with functional dependencies', *VLDB J.* **21**(1), 121–144.

Hartmann, S. & Link, S. (2012), 'The implication problem of data dependencies over SQL table definitions: Axiomatic, algorithmic and logical characterizations', *ACM Trans. Database Syst.* **37**(2), 13.

Huhtala, Y., Kärkkäinen, J., Porkka, P. & Toivonen, H. (1999), 'TANE: An efficient algorithm for discovering functional and approximate dependencies', *The Computer Journal* **42**(2), 100–111.

Köhler, H. & Link, S. (2010), 'Armstrong axioms and Boyce-Codd-Heath normal form under bag semantics', *Inf. Process. Lett.* **110**(16), 717–724.

Kolahi, S. (2007), 'Dependency-preserving normalization of relational and XML data', *J. Comput. Syst. Sci.* **73**(4), 636–647.

Lamperti, G., Melchiori, M. & Zanella, M. (2000), On multisets in database systems, *in* 'Proceedings of WOMP', number 2235 *in* 'Lecture Notes in Computer Science', pp. 147–216.

Langeveldt, W.-D. & Link, S. (2010), 'Empirical evidence for the usefulness of Armstrong relations in the acquisition of meaningful functional dependencies', *Inf. Syst.* **35**(3), 352–374.

Link, S. (2012), Armstrong databases: Validation, communication and consolidation of conceptual models with perfect test data, *in* A. Ghose & F. Ferrarotti, eds, 'Asia-Pacific Conference on Conceptual Modelling (APCCM 2012)', Vol. 130 of *CRPIT*, ACS, Melbourne, Australia, pp. 3–20.

Liu, J., Li, J., Liu, C. & Chen, Y. (2012), 'Discover dependencies from data - a review', *IEEE Trans. Knowl. Data Eng.* **24**(2), 251–264.

Maier, D. (1980), 'Minimum covers in relational database model', *J. ACM* **27**(4), 664–674.

Mannila, H. & Räihä, K.-J. (1986), 'Design by example: An application of Armstrong relations', *J. Comput. Syst. Sci.* **33**(2), 126–141.

Mannila, H. & Räihä, K.-J. (1992), 'On the complexity of inferring functional dependencies', *Discrete Applied Mathematics* **40**(2), 237–243.

Mannila, H. & Räihä, K.-J. (1994), 'Algorithms for inferring functional dependencies from relations', *Data Knowl. Eng.* **12**(1), 83–99.

Naumann, F. & Herschel, M. (2010), *An Introduction to Duplicate Detection*, Synthesis Lectures on Data Management, Morgan & Claypool Publishers.

Thalheim, B. (1989), 'On semantic issues connected with keys in relational databases permitting null values', *Elektronische Informationsverarbeitung und Kybernetik* **25**(1/2), 11–20.

Thalheim, B. (2000), *Entity-Relationship Modeling: Foundations of Database Technology*, Springer.

Thi, V. (1986), 'Minimal keys and antikeys', *Acta Cybern.* **7**(4), 361–371.

Vincent, M. (1999), 'Semantic foundation of 4NF in relational database design', *Acta Inf.* **36**, 1–41.

Vincent, M., Liu, J. & Liu, C. (2004), 'Strong functional dependencies and their application to normal forms in XML', *ACM Trans. Database Syst.* **29**(3), 445–462.

Weddell, G. (1992), 'Reasoning about functional dependencies generalized for semantic data models', *ACM Trans. Database Syst.* **17**(1), 32–64.