Armstrong databases: validation, communication and consolidation of conceptual models with perfect test data

Sebastian Link

Department of Computer Science,
The University of Auckland,
Auckland, New Zealand,
Email: s.link@auckland.ac.nz

Abstract

Conceptual models are relational database schemata that result from conceptual data modeling. Conceptual models usually capture the semantics of the underlying application domain inadequately. Therefore, the structure of the conceptual model is often inadequate, too. Academic and commercial database design tools often advocate the use of good test data to validate the adequacy of the conceptual models they produce. In this article we provide evidence that Armstrong databases constitute perfect test data. In particular, Armstrong databases capture perfectly the perceptions of the design team about the semantics of the application domain. Therefore, Armstrong databases serve as an excellent medium to validate and consolidate the understanding of an application domain's semantics, and to communicate this understanding between different stakeholders of the target database. An overview is given about recent advancements on the structural and computational properties of Armstrong databases. These advancements suggest that Armstrong databases provide the foundations necessary to establish an agile database design methodology. Such a methodology complements existing approaches to database design, and is not meant to replace them.

Keywords: Agile database design, Armstrong database, Conceptual model, Constraint, Design by example, Implication, SQL, Test data

1 An Agile Database Design Methodology

Quality database schemata must capture both the structure and semantics of the underlying application domain. Conceptual data modeling (UML, ER, Description logics etc.) takes as input a natural language description of the target database and produces a conceptual model, i.e. a first formal approximation of the target database's schema. In most cases the model produced does not address the semantics of the application domain appropriately. There are several potential reasons: the list of requirements was incomplete, requirements were lost or ignored during conceptual modeling, or the requirements were not fully understood, to name a few. Since the structure of the target database depends crucially on the semantics of the application domain, the structure of a conceptual model is usually inappropriate, too. In practice, database design is often completely automated, i.e. conceptual models are neither validated, nor communicated, nor refined. A classical example is the introduction of *identifiers* or *surrogate keys* as an excuse for good database design. Consider a simple database that collects basic information about the weekly schedule of courses. That is, we have a schema Schedule with columns *C_ID*, *L_Name*, *Time* and *Room*. The schema stores the time (including the weekday) and room in which a lecturer (identified by their L_Name) gives a course (identified by their C_ID). In addition, an artificial column with name *ID* is introduced that carries different values in different rows. A small snapshot of an instance is shown below.

ID	*C_ID*	*Time*	*L_Name*	*Room*
id_1	11301	Mon, 10am	Church	Red
id_2	11301	Mon, 10am	Gödel	Green

While the values in *ID* allow database management systems to uniquely identify rows, the column does not carry any meaning nor does it guarantee consistency, nor does it guarantee efficient processing of updates or queries. An improved design for our example would be the SQL table definition

```
CREATE TABLE Schedule (
    C_ID CHAR[5], L_Name VARCHAR,
    Time CHAR[15], Room VARCHAR,
    PRIMARY KEY (C_ID, Time);)
```

Here, the primary key forces rows over Schedule to be `NOT NULL` in the *C_ID* and *Time* columns, and to be unique on their {*C_ID*, *Time*} projections. In contrast to the *ID* column above, the primary key carries significant meaning: no two distinct rows must have the same value in both the *C_ID* column and the *Time* column. Therefore, some meaningless instances such as the one above, where Church and Gödel teach the same course at the same time in different rooms, are prevented from becoming instances of the schema Schedule with the primary key on *C_ID* and *Time*. It is a big challenge for design teams to identify semantically meaningful constraints. These prevent potential database instances that are semantically meaningless from becoming actual database instances. The identification is challenging for several reasons. One reason is a *mismatch in expertise*: domain experts know lots about the domain, but do not know much about databases; while design teams know lots about databases, but do not

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

Table 1: An Armstrong table for SCHEDULE

C_ID	*Time*	*L_Name*	*Room*
11301	Mon, 10am	Church	Red
11301	Tue, 02pm	Church	Red
78200	Mon, 10am	Church	Red
99120	Wed, 04pm	ni	ni

Table 2: Armstrong table for revised constraint set

C_ID	*Time*	*L_Name*	*Room*
11301	Mon, 10am	Church	Red
11301	Tue, 02pm	Church	Red
78200	Mon, 10am	Church	ni
99120	Mon, 10am	ni	Red

know much about the domain. Knowledge transfer is therefore difficult. Domain experts or design teams may even be absent during the design process, or may have opposing views. For the remainder of this paper we assume that there is a design team and there are some domain experts. Leading database design tools, such as *ERWin*, promote the creation of test data to validate, communicate and consolidate the conceptual models they produce (CA Technologies 2011). To illustrate this idea, consider again our running example. Our team of data designers may wonder if the semantics of the application domain has been captured appropriately by the SQL table definition above. They decide to generate some good test data to discuss their current understanding with the domain experts. In fact, a joint inspection of the data sample in Table 1 reveals some concern about the first and third row. In fact, the design team wonders whether the same lecturer can teach different courses at the same time in the same room. The domain experts confirm that this is impossible. As a consequence, the team decides to specify the uniqueness constraint (UC) u(*Time,L_Name,Room*). They produce the data sample in Table 2 to consolidate their new understanding. An inspection of the first and third row, and the first and fourth row, respectively, reveal that the UC u(*Time,L_Name,Room*) should be replaced by the two stronger UCs u(*Time,L_Name*) and u(*Time,Room*). The example shows the potential benefit of inspecting good sample data for the discovery of semantically meaningful SQL keys. As a revised SQL table schema the team specifies

CREATE TABLE SCHEDULE' (
C_ID CHAR[5], L_Name VARCHAR,
Time CHAR[15], Room VARCHAR,
PRIMARY KEY (C_ID, Time),
UNIQUE (L_Name, Time),
UNIQUE (Room, Time);)

The big question remains what exactly constitutes good test data? It will be argued in this article that the concept of *Armstrong databases* can provide perfect test data. Indeed, Armstrong databases are database instances that i) satisfy the constraints currently perceived semantically meaningful by a design team, and ii) violate all the constraints currently perceived semantically meaningless. In this sense, Armstrong databases are perfect representations of the design team's perception of the application domain. The conditions i) and ii) above are unrealistic - in general, it will be impossible to consider

Table 3: Another Armstrong table for SCHEDULE

C_ID	*Time*	*L_Name*	*Room*
11301	Mon, 10am	Church	Red
22413	Mon, 10am	Church	Red
22413	Tue, 12pm	Church	Red
22413	Wed, 02pm	ni	Red
33524	Wed, 02pm	Gödel	Red
33524	Thu, 10am	Gödel	Green
44635	Thu, 12pm	Gödel	ni

all classes of constraints. Database theory knows of at least 100 different classes of constraints, some of these classes are infeasible or, at least, intractable to reason with. It is therefore good practice, and more realistic, to focus on classes of constraints that can express important properties of application domains and can be reasoned with efficiently. For this reason, the notion of an Armstrong database is usually constrained by the context $\mathcal{C}$ which represents the class of database constraints considered. Therefore, an Armstrong database for a finite set Σ of constraints in class $\mathcal{C}$ is a database instance db_Σ that i) satisfies Σ, and ii) violates every constraint in $\mathcal{C}$ that is not implied by Σ. Here, Σ denotes the set of constraints in $\mathcal{C}$ currently perceived meaningful by a design team. For example, the Armstrong database in Table 1 satisfies the uniqueness constraints $u(X)$ if and only if $\{\text{C\_ID}, \text{Time}\}$ is a subset of X; and the `NOT NULL` constraint $nfs(X)$ if and only if $X \subseteq \{\text{C\_ID}, \text{Time}\}$. The concept of an Armstrong database appears to suggest the agile database design methodology illustrated in Figure 1. Before a conceptual model is transformed into the final logical model, the adequacy of its structure $\mathfrak{D}$ and semantics Σ to model the application domain is first validated and consolidated by the design team together with the team of domain experts. For this task, and also to overcome the mismatch in expertise, it is proposed to generate Armstrong databases that perfectly represent the current perceptions of the design team. This will be done continuously until the design team is satisfied with the Armstrong database created. It should be noted here that the class $\mathcal{C}$ imposes a convenient restriction on what semantic knowledge is to be modeled, and therefore on the complexity of the modeling process. The more expressive we choose $\mathcal{C}$ to be, the more difficult it will become to reason with sets Σ of constraints in $\mathcal{C}$, and the more complex we may expect structural and computational properties of Armstrong databases for $\mathcal{C}$ to be. For example, the instance in Table 1 is Armstrong for the set Σ with the uniqueness constraint $u(\text{C\_ID}, \text{Time})$ and the `NOT NULL` constraint $nfs(\text{C\_ID}, \text{Time})$, with respect to the class $\mathcal{C}$ of uniqueness and `NOT NULL` constraints. However, it is not Armstrong if we add functional dependencies to the class $\mathcal{C}$. In this case, an Armstrong database is illustrated in Table 3. Intuitively, it contains more rows than the Armstrong database in Table 1 since the set of potentially meaningful constraints has increased. For example, the third and fourth row violate the functional dependency that the *C_ID* determines the *L_Name*, i.e., under the current perceptions a course may be taught by different lecturers. If it is a semantically meaningful constraint that the same course must be taught by the same lecturer, then the Armstrong database in Table 3 reveals that this semantically meaningful constraint has not been captured by the current perceptions of the design team about the application domain. However,

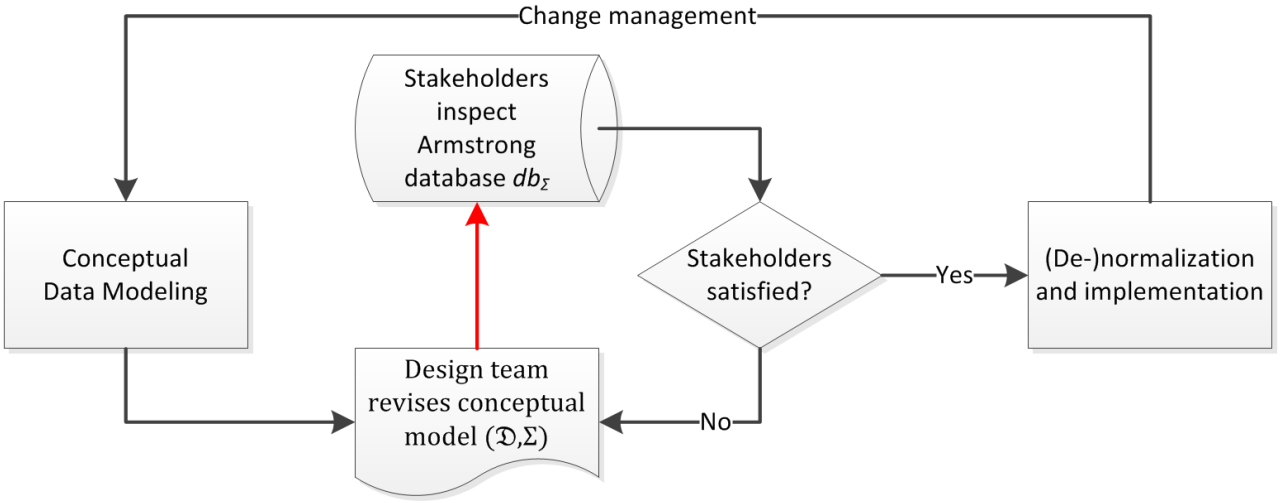

Figure 1: Armstrong databases: foundations for an agile database design methodology

the Armstrong database in Table 1 does not reveal this fact, nor is it designed to do so.

In this article we give a brief overview of recent advances regarding the understanding of structural and computational properties of Armstrong databases for several classes of database constraints. The advances show that Armstrong databases may indeed provide a foundation for an agile methodology to database design, as illustrated in Figure 1. We will answer questions whether Armstrong databases exist, how their structure looks like and how they can be generated when they do exist, and what the time and space complexity is to produce them. That is, we study the area marked red in Figure 1 by providing insights on the feasibility of the methodology proposed.

Organization. We briefly review work on Armstrong databases in the relational model of data in Section 2, and argue that Armstrong databases must also take into account features that can occur in real database instances, e.g. SQL databases. We introduce an SQL-like data model in Section 3. This includes the structure of the model, as well as the semantics modeled by several classes of constraints, i.e. `NOT NULL` and uniqueness constraints, functional dependencies and also cardinality constraints. In subsequent sections we summarize structural and computational properties of Armstrong databases for these classes of database constraints, gradually extending their expressiveness. The increase in expressiveness means, in particular, that new concepts will be introduced with each class analyzed in order to establish the properties of Armstrong databases. In Section 4 we begin our analysis with the class of `NOT NULL` and uniqueness constraints. Subsequently, we add the class of functional dependencies in Section 5. Finally, we add the class of cardinality constraints in Section 6. In Section 7 we conclude and discuss future work.

2 Previous work and new challenges

Data dependencies and Armstrong databases have been studied thoroughly in the relational model of data, cf. (Abiteboul et al. 1995, Fagin 1982*a*). Dependencies are essential to the design of the target database, the maintenance of the database during its lifetime, and all major data processing tasks (Abiteboul et al. 1995, Thalheim 2000). In this section we will briefly summarize the work on Armstrong databases in the relational model of data. Subsequently, we will comment on new challenges arising from features present in SQL databases.

2.1 Armstrong relations

Armstrong relations 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, Katona & Tichler 2006, Hartmann, Leck & Link 2011) and functional dependencies (FDs) (Armstrong 1974, Beeri et al. 1984, Demetrovics & Thi 1995*b*, Mannila & Räihä 1986). Note that in the relational model of data, the class of keys is subsumed by the class of FDs. The existence of Armstrong relations for FDs was shown by Armstrong (Armstrong 1974), and Fagin (Fagin 1982*b*) has shown the existence of Armstrong relations for a large class of data dependencies; however, classes of data dependencies do not necessarily enjoy Armstrong relations by any means. The structure of Armstrong relations for the class of FDs over total relations has mainly been investigated by Beeri, Fagin, Statman and Dowd (Beeri et al. 1984), and Mannila and Räihä (Mannila & Räihä 1986). In the current article, we will summarize extensions of several of their results from the special case of relations to SQL database instances. The properties of Armstrong relations have also been analyzed for various other classes of data dependencies (De Marchi & Petit 2007, Demetrovics & Thi 1995*a*, Fagin 1982*a*, Fagin & Vardi 1983, Gottlob & Libkin 1990, Mannila & Räihä 1986). Cardinality constraints, in particular, have been investigated in conceptual models under a relational semantics (Hartmann 2001, 2003, Lenzerini & Nobili 1990, Liddle et al. 1993, Thalheim 1992), and recently in XML (Ferrarotti et al. 2011, Hartmann & Link 2010*a*). An excellent survey on Armstrong databases is (Fagin 1982*a*). Over the last decades, the concept of Armstrong relations has been found useful in many database applications. *Design aids* (De Marchi et al. 2003, Mannila & Räihä 1986, Silva & Melkanoff 1979) have been developed that utilize Armstrong databases to help judge, justify, convey, or test the design team's understanding of a given application domain. Recently, empirical evidence has been established that Armstrong relations help design teams to recognize semantically meaningful FDs that they were unable to recognize without the help of Armstrong relations (Langeveldt

& Link 2010). Failure to identify some meaningful functional dependency also means that the output of a *requirements analysis* is afflicted with errors. Empirical studies show that more than half the errors which occur during systems development are requirements errors (Enders & Romback 2003, Martin 1989). Requirements errors are also the most common cause of failure in systems development projects (Enders & Romback 2003, Standish Group 1995). The cost of errors increases exponentially over the development life cycle: it is more than 100 times more costly to correct a defect post-implementation than it is to correct it during requirements analysis (Boehm 1981). This suggests that it would be more effective to concentrate quality assurance efforts in the requirements analysis stage, in order to catch requirements errors as soon as they occur, or to prevent them from occurring altogether (Zultner 1992). Hence, there are also strong economic incentives to utilize Armstrong relations for the acquisition of meaningful FDs. Finally, Kolaitis et al. have established first correspondences between unique characterizations of *schema mappings* and the existence of Armstrong bases (Alexe et al. 2010). An Armstrong basis refers to a finite set of pairs consisting of a source instance and a target instance that is a universal solution for the source instance.

2.2 State of database practice

Today, the database industry is worth an estimated 32 billion US dollars, and still growing in the double digits (Meijer & Bierman 2011). Database systems are available from vendors such as Oracle, IBM and Microsoft, and as open-source software including MySQL, PostgreSQL and Ingres. Most data are managed by database systems that adopt the ISO and ANSI industry standard for defining and querying data, i.e. the Structured Query Language (SQL) (Date 2009). SQL was developed by IBM (Chamberlin & Boyce 1974) to implement Edgar Codd's model of relations (Codd 1970). After almost 40 years in use, SQL-based database systems still dominate the market today and influence new paradigms as the field evolves (Anthes 2010). Web models, e.g. XML and RDF, are applied primarily to roll-out, exchange and integrate data that are commonly SQL-based (Elmasri & Navathe 2010). Many websites, e.g. Facebook, and distributed applications, e.g. e-commerce, require high scalability. However, their core data stores and services remain SQL-based (Rys 2011). While SQL covers Codd's model of relations, additional features are meant to ease data processing (Date 2009). SQL data are laid out in tables. These may contain i) duplicate rows to avoid expensive duplicate removal, and ii) null marker occurrences to accommodate partial information in cells of columns declared `NULL`. Relations are tables with no duplicate rows and no null marker occurrences.

2.3 State of database theory

Codd introduced the data model of relations in his seminal paper in 1970 (Codd 1970). His overall achievement was to make data management into a science. In 1981 Codd received the Turing Award "for his fundamental and continuing contributions to the theory and practice of database management systems". Mainstream research has addressed the above challenges in the model of relations (Fagin & Vardi 1986). Around 100 different classes of constraints on relations have been investigated (Thalheim 1991), but few of them are motivated by practice. Research has shown that expressive and tractable classes of constraints can enable quality data management in theory (Abiteboul et al. 1995, Levene & Loizou 1999, Maier 1983, Mannila & Räihä 1992, Ullman 1988). Some researchers have examined the effect of null markers on on the theory of some classes of constraints (Atzeni & Morfuni 1986, Imielinski & Lipski Jr. 1984, Levene & Loizou 1999, Lien 1982). Until very recently (Hartmann & Link 2010*b*), there is no single work in the literature that analyzes constraints on general SQL data. Instead, the community attempts to understand constraints on web data (Akhtar et al. 2010, Arenas & Libkin 2005, 2004, Bojanczyk et al. 2009, Fan & Siméon 2003, Buneman et al. 2002, Hartmann & Link 2010*a*, 2009, Lausen et al. 2008, Vincent et al. 2004). An investigation of constraints on SQL data would not just address the dominant data format in practice, but also more sophisticated formats, such as web data.

2.4 State of disparity

We observe: 1) In practice, SQL data is the premier data format, and 2) In theory, constraints on SQL data have been ignored. Thus, the current knowledge on Armstrong databases applies to only very special instances of SQL data. This distinct disparity between theory and practice restrains the feasibility of the design methodology illustrated in Figure 1.

3 Data model

Based on the presence of duplicate and partial information in SQL database instance, we introduce an SQL-like data model in this section. The model comprises a structural part, as well as a semantical part. For the latter we focus on three types of integrity constraints, i.e. `NOT NULL` and uniqueness constraints, functional dependencies and cardinality constraints. Finally, we recall the formal notion of an Armstrong database.

3.1 Structure

Let $\mathfrak{H} = \{H_1, H_2, \ldots\}$ be a countably infinite set of symbols, called *column headers* or *headers* for short. A *table schema* is a finite non-empty subset T of $\mathfrak{H}$. Each header H of a table schema T is associated with a countably infinite domain $dom(H)$ of the possible values that can occur in column H. To encompass partial information every column can have a null marker, denoted by $\texttt{ni} \in dom(H)$. The intention of `ni` is to mean "no information". We would like to stress that a null marker is different from a domain value. The inclusion of `ni` into the domain is a syntactic convenience.

For header sets X and Y we may write XY for $X \cup Y$. If $X = \{H_1, \ldots, H_m\}$, then we may write $H_1 \cdots H_m$ for X. In particular, we may write simply H to represent the singleton $\{H\}$. A *row* over T (T-row or simply row, if T is understood) is a function $r : T \to \bigcup_{H \in T} dom(H)$ with $r(H) \in dom(H)$ for all $H \in T$. The null marker occurrence $r(H) = \texttt{ni}$ associated with a header H in a row r means that there is no information about $r(H)$. That is, $r(H)$ may not exist or $r(H)$ exists but is unknown. For $X \subseteq T$ let $r(X)$ denote the restriction of the row r over T to X. A *table* t over T is a finite multi-set (bag) of rows over T. For a row r over T and a set $X \subseteq T$, r is said to be X-total if for all $H \in X$, $r(H) \neq \texttt{ni}$. Similar, a table t over T is said to be

X-total, if every row r of t is X-total. A table t over T is said to be a *total table* if it is T-total.

3.2 Semantics

Following Atzeni and Morfuni (Atzeni & Morfuni 1986) a *null-free subschema* (NFS) over the table schema T is a an expression $nfs(T_s)$ where $T_s \subseteq T$. The NFS T_s over T is satisfied by a table t over T, denoted by $\models_t nfs(T_s)$, if and only if t is T_s-total. SQL allows the specification of column headers as `NOT NULL`. NFSs occur in everyday database practice: the set of headers declared `NOT NULL` forms an NFS over the underlying table schema.

Following Lien (Lien 1982) a *functional dependency* (FD) over the table schema T is a statement $X \to Y$ where $X, Y \subseteq T$. The FD $X \to Y$ over T is satisfied by a table t over T, denoted by $\models_t X \to Y$, if and only if for all $r_1, r_2 \in t$ the following holds: if $r_1(X) = r_2(X)$ and r_1, r_2 are X-total, then $r_1(Y) = r_2(Y)$. FDs of the form $\emptyset \to Y$ are called *non-standard*, otherwise FDs are called *standard*. The size $|\sigma|$ of an FD $\sigma = X \to Y$ is defined as $|X| + |Y|$.

We now introduce the concept of a cardinality constraint into databases with partial information. A *cardinality constraint* (CC) over the table schema T is a statement $card(X) \leq b$ where $X \subseteq T$ and b is a positive integer. The CC $card(X) \leq b$ over T is satisfied by a table t over T, denoted by $\models_t card(X) \leq b$, if and only if for all $r_1, r_2, \ldots, r_{b+1} \in t$ the following holds: if $\forall i, j \in \{1, \ldots, b+1\}(r_i(X) = r_j(X))$ and $\forall i \in \{1, \ldots, b+1\}(r_i(X)$ is X-total), then $\exists i, j \in \{1, \ldots, b+1\}(r_i = r_j)$. CCs of the form $card(\emptyset) \leq b$ are called *non-standard*, otherwise CCs are called *standard*. CCs subsume the concept of *uniqueness constraints* for the special case where $card(X) \leq 1$. When we speak of uniqueness constraints (UCs) we use the notation $u(X)$ instead of $card(X) \leq 1$. The size $|\sigma|$ of a CC $\sigma = card(X) \leq b$ is defined as $|X| + \log b$.

For a set Σ of constraints over some table schema T, we say that a table t over T *satisfies* Σ, denoted by $\models_t \Sigma$, if t satisfies every element of Σ. If for some $\sigma \in \Sigma$ the table t does not satisfy σ we sometimes say that t violates σ (in which case t also violates Σ) and write $\not\models_t \sigma$ ($\not\models_t \Sigma$). The size $||\Sigma||$ of a set Σ of FDs and CCs is defined as the sum of sizes over all elements of Σ. The cardinality $|\Sigma|$ of a finite set Σ is defined as the number of its elements.

Example 1 *The SQL table definition* SCHEDULE *from the first section can be captured in our data model as follows. The table schema* $T =$ SCHEDULE *consists of the column headers C_ID, Time, L_Name and Room. The NFS* $nfs(T_s)$ *is defined by* $T_s = \{C\_ID, Time\}$. *The set* Σ *consists of the UC* $u(C\_ID, Time)$. *The table in Table 1 satisfies* Σ *and other constraints such as* $card(L\_Name, Room) \leq 3$ *and* $C\_ID \to L\_Name$. *The same table violates constraints such as* $u(Time, L\_Name, Room)$, $card(L\_Name, Room) \leq 2$ *and* $C\_ID \to Time$.

3.3 Implication

For the design, maintenance and applications of a relational database, data dependencies are identified as semantic constraints on the relations which are intended to be instances of the database schema. During the design process or lifetime of a database one usually needs to determine further dependencies which are logically implied by the given ones. In line with the literature of database constraints, we restrict our attention to the implication of constraints in some fixed class $\mathcal{C}$.

Let T be a table schema, let $nfs(T_s)$ denote an NFS over T, and let $\Sigma \cup \{\varphi\}$ be a finite set of constraints over T. We say that Σ *implies* φ in the presence of $nfs(T_s)$, denoted by $\Sigma \models_{T_s} \varphi$, if every T_s-total table t over T that satisfies Σ also satisfies φ. If Σ does not imply φ in the presence of $nfs(T_s)$ we may also write $\Sigma \not\models_{T_s} \varphi$. Let the set $\Sigma^*_{T_s} = \{\varphi \mid \Sigma \models_{T_s} \varphi\}$ denote the *semantic closure* of Σ and $nfs(T_s)$.

Example 2 *Consider the set* Σ *with NFS* $nfs(T_s)$ *over table schema* T *from Example 1. Then the following are examples of constraints implied by* Σ *in the presence of* $nfs(T_s)$: $u(C\_ID, Time, L\_Name)$, $card(C\_ID, Time) \leq 2$, *and* $C\_ID, Time \to L\_Name$. *However, neither of the constraints* $u(Time, L\_Name)$, $card(L\_Name, Room) \leq 2$ *nor* $C\_ID \to Time$ *are implied by* Σ *in the presence of* $nfs(T_s)$. *Indeed, the table in Table 1 provides a counter-example table for the implication of all of these constraints.*

3.4 Armstrong databases

The formal concept of an *Armstrong database* was originally introduced by Fagin (Fagin 1982*a*). Intuitively, an Armstrong table satisfies the given constraints and violates the constraints in the given class that are not implied by the given constraints. This results in the following definition.

Let $\mathcal{C}$ denote a class of data dependencies. Let Σ be a finite set of elements from $\mathcal{C}$, and $nfs(T_s)$ an NFS over table schema T. A table t over T is said to be $\mathcal{C}$-*Armstrong* for Σ and $nfs(T_s)$ if and only if i) t satisfies Σ, ii) t violates every $\varphi \in \mathcal{C}$ where $\varphi \notin \Sigma^*_{T_s}$, iii) t is T_s-total, and iv) for every $H \in T - T_s$, t is not H-total.

Example 3 *The sample in Table 1 forms a* $\mathcal{C}$*-Armstrong table for* $\Sigma = \{u(C\_ID, L\_Name)\}$ *and* $nfs(C\_ID, L\_Name))$, *where* $\mathcal{C}$ *denotes the class of uniqueness constraints. When* $\mathcal{C}$ *denotes the class of UCs and FDs, then the sample in Table 1 does not form a* $\mathcal{C}$*-Armstrong table for* $\Sigma = \{u(C\_ID, L\_Name)\}$ *and* $nfs(C\_ID, L\_Name)$. *Indeed, the sample satisfies the FD* $C\_ID \to L\_Name$ *even though it is not implied by* Σ *in the presence of* $nfs(C\_ID, L\_Name))$. *The sample in Table 3 is a* $\mathcal{C}$*-Armstrong table for* $\Sigma = \{u(C\_ID, L\_Name)\}$ *and* $nfs(C\_ID, L\_Name))$ *when* $\mathcal{C}$ *denotes the class of UCs and FDs. However, when* $\mathcal{C}$ *denotes the class of CCs and FDs, then the sample in Table 3 is not a* $\mathcal{C}$*-Armstrong table for* $\Sigma = \{u(C\_ID, L\_Name)\}$ *and* $nfs(C\_ID, L\_Name))$. *Indeed, the sample satisfies the CC* $card(C\_ID) \leq 3$ *which is not implied by* Σ *in the presence of* $nfs(C\_ID, L\_Name))$. *In fact, there is no table that is Armstrong for* Σ *and* $nfs(C\_ID, L\_Name))$ *with respect to the class of CCs and FDs.*

4 The class of keys

The goal of this section is to summarize structural and computational properties of Armstrong databases for the class $\mathcal{C}$ of uniqueness constraint in the presence of an NFS.

4.1 Important concepts

A natural question to ask is how we can characterize the structure of tables that are Armstrong for uniqueness constraints in the presence of an NFS. For

this purpose we introduce the formal notion of strong agree sets for pairs of distinct rows, and tables.

4.1.1 Strong agree sets

For two rows r_1 and r_2 over table schema T where $r_1 \neq r_2$ we define the *strong agree set* of r_1 and r_2 as the set of all column headers over T on which r_1 and r_2 have the same total value, i.e., $ag^s(r_1, r_2) = \{H \in T \mid r_1(H) = r_2(H) \text{ and } r_1(H) \neq \texttt{ni} \neq r_2(H)\}$. Furthermore, the strong agree set of a table t over table schema T is defined as $ag^s(t) = \{ag^s(r_1, r_2) \mid r_1, r_2 \in t \wedge r_1 \neq r_2\}$.

Example 4 *Let* SCHEDULE $= CTLR$ *with* SCHEDULE$_s$ $= CT$. *Let* Σ *consist of the two UCs* $u(CT)$ *and* $u(LTR)$. *Let* t *denote the data sample in Table 2. The strong agree set of* t *consists of* CLR, LT, TR, L, R, *and* T.

4.1.2 Anti-keys

For a table t to be Armstrong for Σ and $nfs(T_s)$, t must violate all uniqueness constraints $u(X)$ not implied by Σ in the presence of $nfs(T_s)$. Instead of violating all uniqueness constraints it suffices to violate those ones that are maximal with the property that they are not implied by Σ in the presence of $nfs(T_s)$. This motivates the following definition.

Let Σ be a set of UCs, and $nfs(T_s)$ an NFS over table schema T. The set Σ^{-1} of all *anti-keys* with respect to Σ and $nfs(T_s)$ is defined as $\Sigma^{-1} = \{a(X) \mid X \subseteq T \wedge \Sigma \not\models_{T_s} u(X) \wedge \forall H \in (T - X)(\Sigma \models_{T_s} u(XH))\}$. Hence, an anti-key for Σ is given by a maximal set of column headers which does not form a uniqueness constraint implied by Σ.

Example 5 *Let* SCHEDULE $= CTLR$ *with* SCHEDULE$_s$ $= CT$. *Let* Σ *consist of the two UCs* $u(CT)$ *and* $u(LTR)$. *The set* Σ^{-1} *of anti-keys with respect to* Σ *and* SCHEDULE$_s$ *consists of* $a(CLR)$, $a(LT)$ *and* $a(TR)$.

4.2 Structural properties

The concepts from the last sub-section are sufficient to establish a characterization of Armstrong tables for the class of UCs over SQL table schemata.

Theorem 1 *Let* Σ *a set of UCs, and* $nfs(T_s)$ *an NFS over the table schema* T. *For all tables* t *over* T *it holds that* t *is an Armstrong table for* Σ *and* $nfs(T_s)$ *if and only if the following three conditions are satisfied:*

1. $\forall a(X) \in \Sigma^{-1}(X \in ag^s(t))$,
2. $\forall u(X) \in \Sigma \forall Y \in ag^s(t)(X \not\subseteq Y)$,
3. $total(t) = \{H \in T \mid \forall r \in t(r(H) \neq \texttt{ni})\} = T_s$.

Example 6 *Let* SCHEDULE $= CTLR$ *with* SCHEDULE$_s$ $= CT$. *Let* Σ *consist of the two UCs* $u(CT)$ *and* $u(LTR)$, *and let* t *denote the sample on the left of Table 2. Recall from the previous examples that* $\Sigma^{-1} = \{a(CLR), a(LT), a(TR)\}$, *and* $ag^s(t) = \{CLR, LT, TR, L, R, T\}$. *Since* t *satisfies the three conditions of Theorem 1, it follows that* t *is an Armstrong table for* Σ *and* SCHEDULE$_s$.

4.3 Computational properties

We will now use the characterization of Theorem 1 to compute Armstrong tables for an arbitrarily given set Σ of UCs and an arbitrarily given NFS $nfs(T_s)$ over an arbitrarily given table schema T. A great part of the computation is devoted to determine the set Σ^{-1}. For this purpose, we borrow concepts from hyper-graphs. Indeed, to compute Σ^{-1} we generate the simple hyper-graph $\mathcal{H} = (V, E)$ with vertex set $V = T$ and the set $E = \{X \mid u(X) \in \Sigma\}$ of hyper-edges. In fact, we can assume without loss of generality that Σ consists of minimal UCs only. That is, if $u(X)$ and $u(Y)$ are both in Σ, then $X \not\subseteq Y$ and $Y \not\subseteq X$. If not, then we remove all those UCs from Σ that are not minimal, i.e. all those $u(Y) \in \Sigma$ such that there is some $u(X) \in \Sigma$ where $X \subset Y$. From this we obtain $\Sigma^{-1} = \{a(T - X) \mid X \in Tr(\mathcal{H})\}$ where $Tr(\mathcal{H})$ denotes the minimal transversals of the hyper-graph $\mathcal{H}$, i.e. the set of minimal sets X of T that have a non-empty intersection with each hyper-edge of $\mathcal{H}$ (Eiter & Gottlob 1995).

Lemma 1 *Let* Σ *be a set of UCs over table schema* T. *Then* $\Sigma^{-1} = \{a(T - X) \mid X \in Tr(\mathcal{H})\}$.

We have now a complete strategy for computing Armstrong tables. That is, we first compute the set of anti-keys, and then produce rows whose strong agree sets match these anti-keys. The algorithm can also handle the special case where Σ contains the empty key $u(\emptyset)$, saying that each table can have at most one row. This case is dealt with in step $(A0)$. The final step $(A4)$ introduces null marker occurrences in columns that do not belong to the NFS.

Algorithm 2 (Armstrong table computation)

Input: $(T, \Sigma, nfs(T_s))$ with

- a set Σ of UCs, and NFS $nfs(T_s)$ over T;

Output: Armstrong table t over T for Σ and $nfs(T_s)$

Method: let $c_{H,0}, c_{H,1}, \ldots \in dom(H)$ be distinct

(A0) `if` $u(\emptyset) \in \Sigma$ `then return`
$t := \{r_0\}$ where $\forall H \in T$
$r_0(H) := \begin{cases} c_{H,0} & \text{, if } H \in T_s \\ \texttt{ni} & \text{, else} \end{cases}$;
`else` $t := \{r_0\}$ where $\forall H \in T$
$r_0(H) = \{c_{H,0}\}$
`endif`;

(A1) `compute` Σ^{-1} using hypergraph methods;

(A2) $i := 1$;

(A3) `for all` $Y \in \Sigma^{-1}$ `do`
$t := t \cup \{r_i\}$ `where`
$r_i(H) := \begin{cases} c_{H,0} & \text{, if } H \in Y \\ c_{H,i} & \text{, if } H \in T_s - Y \\ \texttt{ni} & \text{, else} \end{cases}$;
$i := i + 1$;
`endfor`;

(A4) $total(t) := \{H \in T \mid \forall r \in t(r(H) \neq \texttt{ni})\}$;
`if` $total(t) - T_s \neq \emptyset$, `then return`
$t := t \cup \{r_i\}$ where $\forall H \in T$
$r_i(H) := \begin{cases} \texttt{ni} & \text{, if } H \in total(t) - T_s \\ c_{H,i} & \text{, else} \end{cases}$
`else return` t `endif`;

Example 7 *Let* SCHEDULE = *CTLR with* SCHEDULE_s = *CT. Let* Σ *consist of the two UCs* $u(CT)$ *and* $u(LTR)$. *On input* $(\text{SCHEDULE}, \Sigma, \text{SCHEDULE}_s)$, *Algorithm 1 would compute the following Armstrong table:*

C_ID	*Time*	*L_Name*	*Room*
$c_{H,0}$	$c_{T,0}$	$c_{L,0}$	$c_{R,0}$
$c_{H,0}$	$c_{T,1}$	$c_{L,0}$	$c_{R,0}$
$c_{H,1}$	$c_{T,0}$	$c_{L,0}$	***ni***
$c_{H,2}$	$c_{T,0}$	***ni***	$c_{R,0}$

A suitable value substitution yields the sample in Table 2.

The following result follows directly from Lemma 1 and Theorem 1.

Theorem 3 *On input* $(T, \Sigma, nfs(T_s))$, *Algorithm 2 computes a table* t *over* T *that is Armstrong for* Σ *and* $nfs(T_s)$.

4.4 Complexity considerations

In this subsection, we investigate properties regarding the space and time complexity for the computation of Armstrong tables. We recall what we mean by *precisely exponential* (Beeri et al. 1984). Firstly, it means that there is an algorithm for computing an Armstrong table, given a set Σ of data dependencies and an NFS $nfs(T_s)$, where the running time of the algorithm is exponential in $||\Sigma||$. Secondly, it means that there is a set Σ of data dependencies and an NFS $nfs(T_s)$ in which the number of rows in each minimum-sized Armstrong table for Σ and $nfs(T_s)$ is exponential in $||\Sigma||$ — thus, an exponential amount of time is required in this case simply to write down the table.

Proposition 1 *The complexity of finding an Armstrong table, given a set of UCs and an NFS, is precisely exponential in the size of the given constraints.*

4.4.1 The size of Armstrong tables

Despite the general worst-case exponential complexity in the size of Σ, Algorithm 2 is a fairly simple algorithm that is, as we show now, quite conservative in its use of time. Let the size of an Armstrong table be defined as the number of rows that it contains. It is a practical question to ask how many rows a minimum-sized Armstrong table requires. An Armstrong table t for Σ and $nfs(T_s)$ is said to be *minimum-sized* if there is no Armstrong table t' for Σ and $nfs(T_s)$ such that $|t'| < |t|$. That is, for a minimum-sized Armstrong table for Σ and $nfs(T_s)$ there is no Armstrong table for Σ and $nfs(T_s)$ with a smaller number of rows.

Proposition 2 *Let* Σ *be a set of UCs,* $nfs(T_s)$ *an NFS over table schema* T. *Let* t *be a minimum-sized Armstrong table for* Σ *and* $nfs(T_s)$. *Then* $\frac{\sqrt{1 + 8 \cdot |\Sigma^{-1}|}}{2} \leq |t| \leq |\Sigma^{-1}| + 2$.

Note the upper bound in Proposition 2. In general, the focus on UCs can yield Armstrong tables with a substantially smaller number of rows than Armstrong tables for more expressive classes such as functional dependencies. The reason is that we do not need to violate any functional dependencies not implied by the given set. In practice, this is desirable for the validation of schemata known to be in Boyce-Codd normal form, for example. Such schemata are often the result of Entity-Relationship modeling. More generally, we can conclude that Algorithm 2 always computes an Armstrong table of reasonably small size.

Theorem 4 *On input* $(T, \Sigma, nfs(T_s))$, *Algorithm 2 computes an Armstrong table for* Σ *and* $nfs(T_s)$ *whose size is at most quadratic in the size of a minimum-sized Armstrong table for* Σ *and* $nfs(T_s)$.

4.4.2 Size of representations

We show that, in general, there is no most concise way of representing the information inherent in a set of UCs and a null-free subschema.

Theorem 5 *There is some set* Σ *of UCs and an NFS* $nfs(T_s)$ *such that* Σ *has size* $\mathcal{O}(n)$, *and the size of a minimum-sized Armstrong table for* Σ *and* $nfs(T_s)$ *is* $\mathcal{O}(2^{n/2})$. *There is some table schema* T, *some NFS* $nfs(T_s)$ *and some set* Σ *of UCs over* T *such that there is an Armstrong table for* Σ *and* $nfs(T_s)$ *where the number of rows is in* $\mathcal{O}(n)$, *and the number of minimal UCs implied by* Σ *in the presence of* $nfs(T_s)$ *is in* $\mathcal{O}(2^n)$.

For the first claim let $T = H_1, \ldots, H_{2n}$, $T_s = T$ and $\Sigma = \{u(H_{2i-1}H_{2i}) \mid i = 1, \ldots, n\}$. Then $\Sigma^{-1} = \{a(X_1 \cdots X_n) \mid \forall i = 1, \ldots, n(X_i \in \{H_{2i-1}, H_{2i}\})\}$. For the second claim let $T = H_1H_1' \cdots H_nH_n'$, $T_s = T$ and $\Sigma = \{u(X_1 \cdots X_n) \mid \forall i = 1, \ldots, n(X_i \in \{H_i, H_i'\})\}$. Then the set of minimal UCs implied by Σ is Σ itself. Since $\Sigma^{-1} = \{a(H_1H_1' \cdots H_{i-1}H_{i-1}'H_{i+1}H_{i+1}' \cdots H_nH_n') \mid i = 1, \ldots, n\}$ there is an Armstrong table for Σ and $nfs(T_s)$ where the number of rows is in $\mathcal{O}(n)$.

We can see that the representation in form of an Armstrong table can offer tremendous space savings over the representation as a UC set, and vice versa. It seems intuitive to use the representation as Armstrong tables for the discovery of semantically meaningful constraints, and the representation as constraint sets for the discovery of semantically meaningless constraints. This intuition has been confirmed empirically for the class of functional dependencies over relations (Langeveldt & Link 2010).

5 Adding functional dependencies

The goal of this section is to summarize structural and computational properties of Armstrong databases for the class $\mathcal{C}$ of UCs and FDs in the presence of an NFS. Note that the potential presence of duplicate rows means, in particular, that the class of FDs does not subsume the class of UCs, in contrast to relations where no duplicate rows can occur. For example, take any table schema and any table with two duplicate rows over the table schema. Then this table satisfies every FD, but violates every UC over the schema.

5.1 FDs do not enjoy Armstrong tables

We will show first that the class of FDs does not enjoy Armstrong tables if there are headers that are not declared `NOT NULL`. Intuitively in such a case, non-standard FDs force all the rows of a table to have the same value on some header. For tables to be Armstrong, however, it may also be required that the values of such a header do differ for some distinct tuples. Consequently, Armstrong tables cannot exist in general.

Theorem 6 *The class of functional dependencies in the presence of a null-free subschema does not enjoy Armstrong tables.*

Let $T = ABC$, $T_s = BC$ and let $\Sigma = \{\emptyset \to A, A \to B\}$. It can be shown formally that there is no Armstrong table for Σ and $nfs(T_s)$. While this is bad news, in general, the negative result stems from the permission of non-standard FDs. For the class of standard UCs and standard FDs in the presence of an NFS, Armstrong tables always exist.

5.2 Important concepts

First we would like to establish sufficient and necessary conditions when a given table is an Armstrong table with respect to a given set Σ of UCs and FDs and an NFS $nfs(T_s)$. This would generalize a well-known result by Mannila, Räihä, Beeri, Dowd, Fagin and Statman for FDs over total database relations (Beeri et al. 1984, Mannila & Räihä 1986).

5.2.1 Maximal sets

Especially useful in this regard is Mannila and Räihä's notion of maximal sets (Mannila & Räihä 1986) which we generalize here from total relations to tables. Let Σ be a set of CCs and FDs and let $nfs(T_s)$ be an NFS over table schema T. For a column header $H \in T$ we define the *maximal sets* $max_{\Sigma,T_s}(H)$ of H with respect to Σ and $nfs(T_s)$ as follows:

$$max_{\Sigma,T_s}(H) := \{X \subseteq T \mid \Sigma \not\models_{T_s} X \to H \wedge \forall H' \in T - X(\Sigma \models_{T_s} XH' \to H)\}.$$

The *maximal sets* of T with respect to Σ and $nfs(T_s)$ are defined as $max_{\Sigma,T_s}(T) = \bigcup_{H \in T} max_{\Sigma,T_s}(H)$. If Σ and $nfs(T_s)$ are clear from the context we may simply write $max(H)$ and $max(T)$, respectively. Thus, the maximal sets of a column header H with respect to Σ and $nfs(T_s)$ are the maximal header subsets of T that do not functionally determine H.

Example 8 *Consider the table schema* $T =$ SCHEDULE*, NFS* $nfs(T_s)$ *where* $T_s = \{C\_ID, Time\}$*, and*

$$\Sigma = \{u(C\_ID, Time), u(Time, L\_Name), u(Time, Room), C\_ID \to L\_Name\}.$$

Then the maximal sets for the column headers are:

- $max_{\Sigma,T_s}(C\_ID) = \{\{L\_Name, Room\}, \{Time\}\}$,
- $max_{\Sigma,T_s}(Time) = \{\{C\_ID, L\_Name, Room\}\}$,
- $max_{\Sigma,T_s}(L\_Name) = \{\{Time\}, \{Room\}\}$,
- $max_{\Sigma,T_s}(Room) = \{\{C\_ID, L\_Name\}, \{Time\}\}$.

Example 9 *Consider the table schema* CONTACT *with headers Address, City and ZIP, NFS* $nfs(T_s)$ *where* $T_s =$ CONTACT$_s = \{ZIP\}$*, and*

$$\Sigma = \{u(Address, City), ZIP \to City\}.$$

Then the maximal sets for the column headers are:

- $max_{\Sigma,T_s}(Address) = \{\{City, ZIP\}\}$,
- $max_{\Sigma,T_s}(City) = \{\{Address\}\}$,
- $max_{\Sigma,T_s}(ZIP) = \{\{Address\}, \{City\}\}$.

5.2.2 Duplicate sets

It is a necessary condition for an Armstrong table that for each maximal set there must be distinct rows in the table whose strong agree set is the maximal set. This is to guarantee that all the FDs not implied by the set of UCs and FDs in the presence of an NFS are violated. Over tables, however, it is still possible that there are UCs $u(X)$ not implied by Σ in the presence of $nfs(T_s)$ over T, even if the FD $X \to T$ is implied. For this reason we also require of Armstrong tables that for all column header sets X that are maximal with this property there must be distinct rows in the table whose strong agree set is X. This motivates the following definition.

Let Σ be a set of UCs and FDs and let $nfs(T_s)$ be an NFS over table schema T. We define the *duplicate sets* $dup_{\Sigma,T_s}(T)$ of T with respect to Σ and $nfs(T_s)$ as follows:

$$dup_{\Sigma,T_s}(T) := \{X \subseteq T \mid \Sigma \models_{T_s} X \to T \wedge \Sigma \not\models_{T_s} u(X) \wedge \forall H' \in T - X(\Sigma \models_{T_s} u(XH'))\}.$$

If Σ and $nfs(T_s)$ are clear from the context we may simply write $dup(T)$.

Example 10 *Consider the table schema* $T =$ SCHEDULE*, NFS* $nfs(T_s)$ *where* $T_s = \{C\_ID, Time\}$*, and*

$$\Sigma = \{u(C\_ID, Time), u(Time, L\_Name), u(Time, Room), C\_ID \to L\_Name\}.$$

Then $dup_{\Sigma,T_s}($SCHEDULE$) = \emptyset$.

Example 11 *Consider the table schema* CONTACT *with headers Address, City and ZIP, NFS* $nfs(T_s)$ *where* $T_s =$ CONTACT$_s = \{ZIP\}$*, and*

$$\Sigma = \{u(Address, City), ZIP \to City\}.$$

Then $dup_{\Sigma,T_s}($CONTACT$) = \{\{Address, ZIP\}\}$.

5.2.3 Notions of agree sets

For our anticipated characterization of Armstrong tables for UCs and FDs the notion of a strong agree set is insufficient. In fact, we require different notions of agree sets. Let t be a table, and r_1, r_2 be two rows over table schema T. The *agree set* of r_1 and r_2 is defined as $ag(r_1, r_2) = (X, Y)$ where $\forall H \in T(((r_1(H) = r_2(H) \wedge r_1(H) \neq \texttt{ni}) \leftrightarrow H \in X) \wedge (r_1(H) = r_2(H) \leftrightarrow H \in Y))$. The *strong agree set* of r_1 and r_2 is defined as $ag^s(r_1, r_2) = X$ where $ag(r_1, r_2) = (X, Y)$. The *weak agree set* of r_1 and r_2 is defined as $ag^w(r_1, r_2) = Y$ where $ag(r_1, r_2) = (X, Y)$. The *agree set* of t is defined as $ag(t) = \{ag(r_1, r_2) \mid r_1, r_2 \in t \wedge r_1 \neq r_2\}$. The *strong agree set* of t is defined as $ag^s(t) = \{X \mid (X, Y) \in ag(t)\}$. The *weak agree set* of t is defined as $ag^w(t) = \{Y \mid (X, Y) \in ag(t)\}$. For $X \in ag(t)$ we define $w(X) = \bigcap\{Y \mid (X, Y) \in ag(t)\}$. While strong and weak agree sets coincide over total relations, the distinction between the two is crucial for tables.

Example 12 *Let* t *denote the following table over* SCHEDULE $= \{C, T, L, R\}$.

C_ID	*Time*	*L_Name*	*Room*
11301	*Mon, 10am*	*Church*	*Red*
22413	*Tue, 12pm*	*Church*	*Red*
33424	*Tue, 12pm*	*Gödel*	*Green*
33424	*Wed, 02pm*	*Gödel*	*Green*
44535	*Wed, 04pm*	*Turing*	*Green*
44535	*Thu, 10am*	*Turing*	**ni**
55646	*Thu, 02pm*	**ni**	*Yellow*

Then $ag^s(t) = ag^w(t)$ contain LR, $\emptyset$, T, CLR, R and CL.

Example 13 *Let t denote the following table over* CONTACT $= \{A, C, Z\}$.

Address	*City*	*ZIP*
70 King St	***ni***	*10001*
70 King St	*San Francisco*	*94107*
ni	*Manhattan*	*10002*
ni	*Manhattan*	*10003*
03 Astor Pl	*Manhattan*	*10003*
03 Astor Pl	*Morinville*	*10003*

Here, $ag^s(t) = \{A, \emptyset, C, CZ, AZ\}$, and $ag^w(t) = \{A, \emptyset, AC, CZ, AZ\}$. Moreover, $w(C) = AC$.

5.3 Structural properties

These notions allow us to obtain the following characterization of Armstrong tables. For a set Σ of FDs and a set X of column headers over T let $X^*_{\Sigma,T_s} = \{H \in T \mid \Sigma \models_{T_s} X \to H\}$.

Theorem 7 *Let T be a table schema, Σ a set of UCs and FDs, and* $\mathrm{nfs}(T_s)$ *an NFS over T. For all tables t over T it holds that t is an Armstrong table for Σ and* $\mathrm{nfs}(T_s)$ *if and only if all of the following conditions are satisfied:*

1. $\forall H \in T \forall X \in max_{\Sigma,T_s}(H)$ $(X \in ag^s(t) \wedge H \notin w(X))$,
2. $\forall X \in ag^s(t)(X^*_{\Sigma,T_s} \subseteq w(X))$,
3. $\forall X \in dup_{\Sigma,T_s}(T)(X \in ag^s(t))$,
4. $\forall X \in ag^s(t) \forall u(Z) \in \Sigma(Z \not\subseteq X)$,
5. $total(t) = T_s$.

Example 14 *Consider the table schema $T =$* SCHEDULE*, NFS $nfs(T_s)$ where $T_s = \{C\_ID, Time\}$, and*

$$\Sigma = \{u(C\_ID, Time), u(Time, L\_Name), u(Time, Room), C\_ID \to L\_Name\}.$$

Examples 8, 10 and 12 allow us to verify all the conditions of Theorem 7 for the table t in Example 12. Hence, t is indeed an Armstrong table for Σ and $\mathrm{nfs}(T_s)$.

Example 15 *Consider again the table schema* CONTACT *with headers Address, City and ZIP, NFS* $\mathrm{nfs}(T_s)$ *where $T_s =$* $\mathrm{CONTACT}_s = \{ZIP\}$, *and*

$$\Sigma = \{u(Address, City), ZIP \to City\}.$$

Examples 9, 11 and 13 allow us to verify all the conditions of Theorem 7 for the table t in Example 13. Hence, t is indeed an Armstrong table for Σ and $\mathrm{nfs}(T_s)$.

5.4 Computational properties

The following algorithm computes an Armstrong table for an arbitrary set Σ of standard UCs and FDs and an arbitary NFS $nfs(T_s)$. In step (A0) the algorithm computes the family of maximal sets with respect to $\Sigma[\mathrm{FD}] = \{X \to T \mid u(X) \in \Sigma\} \cup \{X \to Y \mid X \to Y \in \Sigma\}$. The computation can be done by Algorithm 8 in (Hartmann, Kirchberg & Link 2011). In step (A1) the algorithm computes the duplicate sets. To compute $dup_{\Sigma,T_s}(T)$ we generate the hyper-graph $\mathcal{H} = (V, E)$ with vertex set $V = T$ and the set

$$E = \{K - T_s \mid u(K) \in \Sigma\}$$

as hyper-edges. From this we obtain $dup_{\Sigma,T_s}(T)$ as

$$dup_{\Sigma,T_s}(T) = \{T - X \mid X \in Tr(\mathcal{H}) \wedge \forall M \in max_{\Sigma[\mathrm{FD}],T_s}(T)(T - X \not\subseteq M)\}$$

where $Tr(\mathcal{H})$ denotes the minimal transversals of the hyper-graph $\mathcal{H}$. Note that when $X \in dup(T)$, then i) $X \notin max(T)$, ii) $Z := \{H \in T \mid X \in max_{\Sigma[\mathrm{FD}],T_s}(H)\} = \emptyset$, and iii) $T_s \subseteq X$. In particular, step (A5) adds to the table rows that strongly agree on the members of $dup_{\Sigma,T_s}(T)$. If $X \in max_{\Sigma[\mathrm{FD}],T_s}(T)$, then Z contains all columns for which X is maximal, and step $(A5)$ adds rows with a strong agree set on X. Finally, an additional row may be required to introduce the null marker `ni` to columns that are specified `NULL`, see step (A8).

Algorithm 8 (Armstrong table computation)

Input: table schema T, a set Σ of standard UCs and FDs, and an NFS $nfs(T_s)$ over T

Output: Armstrong table t for Σ and $nfs(T_s)$

Method: let $c_{H,1}, c_{H,2}, \ldots \in dom(H)$ be distinct

(A0) `for all` $H \in T$ compute $max_{\Sigma[\mathrm{FD}],T_s}(H)$;

(A1) compute $dup_{\Sigma[\mathrm{FD}],T_s}(T)$;

(A2) $r := \emptyset$; $i := 1$;

(A3) `for all` $X \in max_{\Sigma[\mathrm{FD}],T_s}(T) \cup dup(T)$ `do`

(A4) $Z := \{H \in T \mid X \in max_{\Sigma[\mathrm{FD}],T_s}(H)\}$;

(A5) $t := t \cup \{r_i, r_{i+1}\}$ where $\forall H \in T$

$$r_i(H) := \begin{cases} c_{H,i} & \text{, if } H \in XZT_s \\ \texttt{ni} & \text{, else} \end{cases} \text{; and}$$

$$r_{i+1}(H) := \begin{cases} c_{H,i} & \text{, if } H \in X \\ c_{H,i+1} & \text{, if } H \in Z(T_s - X) \\ \texttt{ni} & \text{, else} \end{cases} ;$$

(A6) $i := i + 2$;

(A7) `enddo`;

(A8) $total(t) := \{H \in T \mid \forall r \in t(r[H] \neq \texttt{ni})\}$;
`if` $total(t) - T_s \neq \emptyset$, `then`
`return` $t := t \cup \{r_i\}$ where $\forall H \in T$

$$r_i(H) := \begin{cases} \texttt{ni} & \text{, if } H \in total(t) - T_s \\ c_{H,i} & \text{, else} \end{cases}$$

`else return` r `endif`;

Example 16 *Consider again the table schema* CONTACT *with headers Address, City and ZIP, NFS* $\mathrm{nfs}(T_s)$ *where $T_s =$* $\mathrm{CONTACT}_s = \{ZIP\}$, *and*

$$\Sigma = \{u(Address, City), ZIP \to City\}.$$

Suppose these items form the input to Algorithm 8. Examples 9 and 11 show the maximal and duplicate sets, respectively, computed in steps (A0) and (A1). Algorithm 8 may compute

Address	*City*	*ZIP*
a_1	c_1	z_1
a_2	c_1	z_1
a_3	c_3	z_3
a_3	c_4	z_4
ni	c_5	z_5
ni	c_5	z_6
a_7	**ni**	z_7
a_7	**ni**	z_7

as an Armstrong table for Σ *and* $nfs(T_s)$.

5.5 Complexity considerations

We present some results regarding the time and space complexity required for the representation of constraints using Armstrong tables for UCs and FDs. The first result is very much the same as Proposition 1 for the class of UCs.

Proposition 3 *The complexity of finding an Armstrong table, given a set of FDs and an NFS, is precisely exponential.*

5.5.1 The size of Armstrong tables

Despite the general worst-case exponential complexity in the size of Σ, Algorithm 8 is a fairly simple algorithm that is, as we show now, quite conservative in its use of time. Note the correspondence to Proposition 2 for the class of UCs.

Proposition 4 *Let* Σ *be a set of standard UCs and standard FDs, let* $nfs(T_s)$ *be some NFS over some table schema* T*, and let* t *be a minimum-sized Armstrong table for* Σ *and* $\mathrm{nfs}(T_s)$*. Then*

$$\frac{\sqrt{1 + 8 \cdot |max_{\Sigma,T_s}(T) \cup dup_{\Sigma,T_s}(T)|}}{2} \leq |r|$$

and

$$|r| \leq 2 \times |max_{\Sigma,R_s}(R) \cup dup_{\Sigma,T_s}(T)| + 1.$$

We can conclude that Algorithm 8 always computes an Armstrong table of reasonably small size.

Theorem 9 *On input* (T, Σ, T_s)*, Algorithm 8 computes an Armstrong table for* Σ *and* $\mathrm{nfs}(T_s)$ *whose size is at most quadratic in the size of a minimum-sized Armstrong table for* Σ *and* $\mathrm{nfs}(T_s)$*.*

5.5.2 The size of representations

None of the representations strictly dominates the other. Therefore, both representations should be used together. An optimal cover of a constraint set Σ is a constraint set Σ' which implies the same constraints as Σ and is of minimum size $||\Sigma'||$.

Theorem 10 *There is some table schema* T*, some set* Σ *of FDs and some NFS* $\mathrm{nfs}(T_s)$ *over* T *such that* Σ *has size* $\mathcal{O}(n)$*, and the size of a minimum-sized Armstrong table for* Σ *and* $\mathrm{nfs}(T_s)$ *is* $\mathcal{O}(2^{n/2})$*. There is some table schema* T*, some set* Σ *of FDs and some NFS* $\mathrm{nfs}(T_s)$ *over* T *such that there is an Armstrong table for* Σ *and* $\mathrm{nfs}(T_s)$ *where the number of rows is in* $\mathcal{O}(n)$*, and the optimal cover of* Σ *with respect to* $\mathrm{nfs}(T_s)$ *has size* $\mathcal{O}(2^n)$*.*

Constraint sets can help to identify constraints incorrectly perceived as meaningful, and Armstrong tables can help to identify constraints incorrectly perceived as meaningless (Langeveldt & Link 2010).

5.6 Use case: transforming relational approximations into SQL table definitions

We consider Beeri and Bernstein's famous example that was originally used to show that dependency-preserving Boyce-Codd Normal Form decompositions cannot always be obtained in the relational model of data (Beeri & Bernstein 1979). Here, we apply our toolbox of Armstrong tables to the relational approximation of the application domain to derive a concise SQL table definition.

Suppose a team of data engineers has decided to use the three columns *Address*, *City* and *ZIP* as part of a contact address management system. They have identified the set Σ with $Address, City \rightarrow ZIP$ and $ZIP \rightarrow City$ as a first approximation for capturing the semantics of the underlying application domain. (This is the example from (Beeri & Bernstein 1979).) Suppose that T_s remains empty for now. Using our toolbox the team produces

Address	*City*	*ZIP*
03 Hudson St	Manhattan	10001
03 Hudson St	Manhattan	10001
70 King St	Manhattan	10001
70 King St	San Francisco	94107
`ni`	San Francisco	94129
15 Maxwell St	`ni`	`ni`

as an Armstrong table for Σ in the presence of $nfs(T_s)$. The domain experts inspect the table. They note from the first two rows that the FD $Address, City \rightarrow ZIP$ should be replaced by the stronger uniqueness constraint $u(Address, City)$. Hence, the revised constraint set becomes: $\Sigma = \{u(Address, City), ZIP \rightarrow City\}$, and T_s remains empty. For these requirements, they produce

Address	*City*	*ZIP*
03 Hudson St	Manhattan	10001
70 King St	Manhattan	10001
70 King St	San Francisco	94107
`ni`	San Francisco	`ni`
15 Maxwell St	`ni`	60609
15 Maxwell St	`ni`	60609

as an Armstrong table for Σ in the presence of $nfs(T_s)$. In particular, $dup_{\Sigma,T_s}(T)$ contains the set $\{Address, ZIP\}$. The domain experts are quite happy with the table, but are confused about the last two rows. After some discussion there is consensus that there is no reason to allow different rows with the same total values on *Address* and *ZIP*. Indeed, the data engineers realize that they have not captured the uniqueness constraint $u(Address, ZIP)$. (It is important to note here that in the relational model the FD $ZIP \rightarrow City$ implies the key $\{Address, ZIP\}$, but over SQL tables the two constraints $u(Address, City)$ and $ZIP \rightarrow City$ do not imply $u(Address, ZIP)$.) For this reason, the team revises the set of constraints again: Σ consist of $u(Address, City)$ and $u(Address, ZIP)$, and $ZIP \rightarrow City$. T_s still remains empty. Then they produce

Address	*City*	*ZIP*
03 Hudson St	Manhattan	10001
70 King St	Manhattan	10001
70 King St	San Francisco	94107
`ni`	San Francisco	`ni`
15 Maxwell St	`ni`	60609

as an Armstrong table for Σ in the presence of $nfs(T_s)$. Here, $dup_{\Sigma,T_s}(T) = \emptyset$. The first two rows ensure that the uniqueness constraint $u(City, ZIP)$ is violated. The domain experts note now that there is

a need to always have precise information in the *Address* and *ZIP* columns. As a consequence, the data engineers decide to include both column headers in the null-free subschema $nfs(T_s)$. This is the same as having the following constraints: $Codd(Address,ZIP)$, $u(Address,City)$ and FD $ZIP \to City$. The table

Address	*City*	*ZIP*
03 Hudson St	Manhattan	10001
70 King St	Manhattan	10001
70 King St	San Francisco	94107
35 Lincoln Blvd	San Francisco	94129
15 Maxwell St	ni	60609

is an Armstrong table for this design choice. One possible SQL table definition is the following:

```
CREATE TABLE Contact (
  Address VARCHAR,
  City VARCHAR,
  ZIP INT,
  UNIQUE(Address,City),
  PRIMARY KEY(Address,ZIP),
  CHECK(Q = 0));
```

The state assertion is based on the following query Q:

```
SELECT COUNT(*)
FROM Contact c1
WHERE c1.ZIP IN (
 SELECT ZIP
 FROM Contact c2
 WHERE c1.ZIP=c2.ZIP
  AND (c1.City <> c2.City
  OR (c1.City IS NULL AND c2.City IS NOT NULL)
  OR (c1.City IS NOT NULL AND c2.City IS NULL)));
```

and can be enforced on the data or middle tier.

6 Adding cardinality constraints

The goal of this section is to summarize structural and computational properties of Armstrong databases for the class $\mathcal{C}$ of CCs and FDs in the presence of an NFS. Note that UCs are cardinality constraints with a fixed upper bound of 1.

6.1 CCs do not enjoy Armstrong tables

It is not difficult to observe that the class of CCs does not enjoy Armstrong tables. Indeed, every table t has only a fixed finite number of rows. That means t satisfies all cardinality constraints where the upper bounds exceed that number of rows. However, such a cardinality constraint cannot always be guaranteed to be implied by the given set of CCs. Consequently, Armstrong tables do not always exist.

Theorem 11 *The class of cardinality constraints in the presence of a null-free subschema does not enjoy Armstrong tables.*

In what follows we will characterize when Armstrong tables do exist, and we will compute Armstrong tables whenever they exist. It can also be decided efficiently in the size of the given constraints whether an Armstrong table exists.

6.2 Important concepts

6.2.1 More notions of agreement

For characterizing the structure of Armstrong tables we have used different notions of agreement between rows of a table. The different notions are motivated by the potential occurrence of null markers on the one hand, and the different classes of constraints we consider on the other hand. For functional dependencies it suffices to compare all pairs of distinct rows. Cardinality constraints, however, require us to compare any finite number of distinct rows, essentially up to the maximum bound that occurs in the given set of constraints.

Let T be a table schema, and t a table over T. For every positive integer $b > 1$ we define $ag^s_b(t) = \{\bigcap_{1 \leq i < j \leq b} ag^s(r_i, r_j) \mid \exists r_1, \ldots, r_b \in t(\forall 1 \leq i < j \leq b(r_i \neq r_j))\}$, $ag^s_1(t) = \{T\}$ and $ag^s_\infty(t) = \emptyset$.

Example 17 *Let t denote the following table over* Schedule.

C_ID	*Time*	*L_Name*	*Room*
c_1	t_1	***ni***	***ni***
⋮	⋮	⋮	⋮
c_5	t_1	***ni***	***ni***
c_6	t_6	***ni***	r_6
⋮	⋮	⋮	⋮
c_{17}	t_{17}	***ni***	r_6
c_{18}	t_{18}	l_{18}	r_{18}
⋮	⋮	⋮	⋮
c_{23}	t_{23}	l_{18}	r_{18}
c_{24}	t_{24}	***ni***	r_{24}
c_{24}	t_{25}	***ni***	r_{24}
c_{24}	t_{26}	***ni***	r_{24}
c_{27}	t_{27}	l_{27}	r_{27}
c_{27}	t_{28}	l_{27}	r_{27}
c_{29}	t_{29}	l_{29}	***ni***
c_{29}	t_{30}	l_{29}	***ni***
c_{31}	t_{31}	l_{31}	r_{31}
c_{32}	t_{32}	l_{32}	r_{31}
c_{33}	t_{33}	l_{33}	r_{33}
c_{34}	t_{33}	l_{34}	r_{34}

Then $ag^s(t) = ag^s_2(t) = \{\emptyset, T, R, LR, CR, CLR, CL\}$, $ag^s_3(t) = \{\emptyset, T, R, LR, CR\}$, $ag^s_4(t) = ag^s_5(t) = \{\emptyset, T, R, LR\}$, $ag^s_6(t) = \{\emptyset, R, LR\}$, $ag^s_7(t) = \cdots = ag^s_{12}(t) = \{\emptyset, R\}$.

6.2.2 Clone clusters

An Armstrong table must violate all the cardinality constraints not implied by the given set. It suffices, however, for any non-empty set X to violate the cardinality constraint $card(X) \leq b_X - 1$ where b_X denotes the minimum positive integer for which $card(X) \leq b_X$ is implied. Moreover, if there are two cardinality constraints $card(X) \leq b_X$ and $card(Y) \leq b_Y$ such that $b_X = b_Y$ and $Y \subseteq X$, then it suffices to violate $card(X) \leq b_X - 1$. This motivates the following definitions.

Let T be a table schema, $nfs(T_s)$ an NFS, and Σ a set of FDs and CCs over T. For $\emptyset \neq X \subseteq T$ let $b_X = \min\{b \mid \Sigma \models_{T_s} card(X) \leq b\}$, if $\exists b \in \mathbb{N}(\Sigma \models_{T_s} card(X) \leq b)$; and $b_X = \infty$, otherwise. The set $cl\text{-}cl_{\Sigma,T_s}(T)$ of *clone clusters* is defined as

$$clcl_{\Sigma,T_s}(T) = \{X \subseteq T \mid \forall H \in T - X(b_{XH} < b_X)\}.$$

For a set Σ of FDs and CCs over table schema T let $\Sigma[\text{FD}] = \{X \to T \mid card(X) \leq 1 \in \Sigma\} \cup \{X \to Y \mid$

$X \to Y \in \Sigma\}$. Note that $b_X = \min\{b \mid \exists card(Y) \leq b \in \Sigma \wedge Y \subseteq XT_s \cap X^*_{\Sigma[\mathrm{FD}],T_s}\}$, if $\exists b \in \mathbb{N}(\Sigma \models_{T_s} card(X) \leq b)$.

Example 18 *Consider the table schema* SCHEDULE, *NFS* $nfs(\mathrm{SCHEDULE}_s)$ *where* $\mathrm{SCHEDULE}_s = CT$, *and*

$$\begin{aligned}\Sigma \;=\; \{&card(C) \leq 3, card(T) \leq 5, card(L) \leq 6,\\ &card(R) \leq 12, card(CT) \leq 1, card(LT) \leq 1,\\ &card(TR) \leq 1, card(CLR) \leq 2, C \to L\}.\end{aligned}$$

Then we have $b_{CTLR} = b_{CTL} = b_{CTR} = b_{TLR} = b_{CT} = b_{LT} = b_{TR} = 1$, $b_{CLR} = b_{CL} = 2$, $b_{CR} = b_C = 3$, $b_T = 5$, $b_{LR} = b_L = 6$, *and* $b_R = 12$. *Therefore,* $cl\text{-}cluster_{\Sigma,\mathrm{SCHEDULE}_s}(\mathrm{SCHEDULE}) = \{T, R, LR, CR, CLR, CTLR\}$.

6.2.3 Essential maximal sets

An Armstrong table must also violate all the functional dependencies not implied by the given set. Recall that it suffices for each column header H to violate all FDs $X \to H$ where X is maximal with the property that $X \to H$ is not implied. Furthermore, if $X \in max_{\Sigma,T_s}(H)$ for some $H \in T$, $X \in cl\text{-}cl_{\Sigma,T_s}(T)$ and X is not maximal for any $H \in T - T_s$, then we can violate $card(X) \leq b_X - 1$ such that for all $H \in T_s - X$, $X \to H$ is also violated. Therefore, we only need to consider *essential maximal sets*, i.e., elements in the following set

$$\begin{aligned}max^{ess}_{\Sigma,T_s}(T) \;=\; &max_{\Sigma,T_s}(T) -\\ &\{X \in cl\text{-}cl_{\Sigma,T_s}(T) \mid\\ &\forall H \in T - T_s (X \notin max_{\Sigma,T_s}(H))\}.\end{aligned}$$

Example 19 *Consider the table schema* SCHEDULE, *NFS* $nfs(\mathrm{SCHEDULE}_s)$ *where* $\mathrm{SCHEDULE}_s = CT$, *and*

$$\begin{aligned}\Sigma \;=\; \{&card(C) \leq 3, card(T) \leq 5, card(L) \leq 6,\\ &card(R) \leq 12, card(CT) \leq 1, card(LT) \leq 1,\\ &card(TR) \leq 1, card(CLR) \leq 2, C \to L\}.\end{aligned}$$

Then

- $max(C) = \{LR, T\}$,
- $max(T) = \{CLR\}$,
- $max(L) = \{T, R\}$, *and*
- $max(R) = \{CL, T\}$.

From $cl\text{-}cluster(\mathrm{SCHEDULE}) = \{T, R, LR, CR, CLR, CTLR\}$ *we conclude that* $max^{ess}(\mathrm{SCHEDULE}) = \{T, R, CL\}$. *Specifically,* $R, T \in cl\text{-}cl(\mathrm{SCHEDULE})$, *but* $R, T \in max(L)$ *and* $L \in \mathrm{SCHEDULE} - \mathrm{SCHEDULE}_s$.

6.3 Structural properties

We are now in a position to establish sufficient and necessary conditions when a given table is Armstrong for a given set Σ of FDs, CCs and an NFS $nfs(T_s)$. In the following theorem, the first (third) condition ensures that all FDs (CCs) not implied by the given set are violated; and the second (fourth) condition ensures that all implied FDs (CCs) are satisfied. The final condition takes into account the NFS.

Theorem 12 *Let T be a table schema, Σ a set of standard FDs and standard CCs, and $nfs(T_s)$ an NFS over T. For all tables t over T it holds that t is an Armstrong table for Σ and $nfs(T_s)$ if and only if all of the following conditions are satisfied:*

1. $\forall H \in T \forall X \in max_{\Sigma,T_s}(H)(X \in ag^s(t) \wedge H \notin w(X))$,
2. $\forall X \in ag^s(t)(X^*_{\Sigma,T_s} \subseteq w(X))$,
3. $\forall X \in cl\text{-}cl_{\Sigma,T_s}(T) \exists Z \in ag^s_{b_X}(t)(X \subseteq Z)$,
4. $\forall card(X) \leq b \in \Sigma \forall Z \in ag^s_{b+1}(t)(X \not\subseteq Z)$,
5. $total(t) = T_s$.

Example 20 *Let* SCHEDULE, $\mathrm{SCHEDULE}_s$ *and* Σ *be as in Example 18. Theorem 12 can be used to verify that the table t from Example 17 is $\mathcal{C}$-Armstrong for Σ and $nfs(\mathrm{SCHEDULE}_s)$ where $\mathcal{C}$ denotes the class of FDs and CCs.*

Theorem 12 will now be applied to compute an Armstrong table for any given set of FDs, CCs and NFS for which an Armstrong table exists.

6.4 Computational properties

For the computation of an Armstrong table, Theorem 12 suggests that the maximal sets and clone clusters need to be computed. The computation of the maximal sets with respect to a set of standard FDs and an NFS $nfs(T_s)$ over table schema T has been studied in (Hartmann, Kirchberg & Link 2011). For the computation of clone clusters we now outline an algorithm that is exponential in the size of Σ. While we leave optimizations of this algorithm for future work, we note that there are sets of FDs and there are sets of CCs, respectively, where every Armstrong table for this set is exponential in the size of Σ, cf. Theorem 5.

Let Σ denote a set of standard FDs and standard CCs, $nfs(T_s)$ an NFS, and X a set of column headers over table schema T. We first compute b_X. We start with $b_X := \infty$ and compute $X^*_{\Sigma[\mathrm{FD}],T_s}$ using Algorithm 1 from (Hartmann, Kirchberg & Link 2011). Then, for each $card(Y) \leq b \in \Sigma$ such that $Y \subseteq XT_s \cap X^*_{\Sigma[\mathrm{FD}],T_s}$ and $b < b_X$ we redefine $b_X := b$. Next we compute the clone clusters in $cl\text{-}cl_{\Sigma,T_s}(T)$. We start with $cl\text{-}cl_{\Sigma,T_s}(T) = \{X \mid \emptyset \neq X \subseteq T\}$. Then for each $X \in cl\text{-}cl_{\Sigma,T_s}(T)$ and each $H \in T - X$ such that $b_{XH} = b_X$ we redefine $cl\text{-}cl_{\Sigma,T_s}(T) := cl\text{-}cl_{\Sigma,T_s}(T) - \{X\}$. Therefore, the time to compute $cl\text{-}cl_{\Sigma,T_s}(T)$ and b_X for each $X \in cl\text{-}cl_{\Sigma,T_s}(T)$ is in $\mathcal{O}(2^{|T|} \times |T| \times ||\Sigma||)$.

Algorithm 13 shows the computation of an Armstrong table. The first two steps consist of the computations of the maximal sets covered in previous work, and the computation of the clone clusters X and their associated b_X. In step (A4), the algorithm generates for each clone cluster X a block of b_X rows that satisfies $card(X) \leq b_X$, violates $card(X) \leq b_X - 1$, and violates every FD $X \to H$ where $H \in T_s - X$. Note that in this case there cannot be any $H \in T_s - X$ such that $X \to H$ is implied by Σ and $nfs(T_s)$. Otherwise, since X is a clone cluster it would hold that $card(XH) \leq b < b_X$ is an implied cardinality constraint. Then, however, $card(X) \leq b < b_X$ would be implied, too. This is a contradiction. In step (A5), the algorithm computes for each maximal set X the set Z of column headers in $T - T_s$ for which X is maximal, and produces two rows which strongly agree on X and disagree on each column header in Z; unless X is also a clone cluster and each column header for which X is maximal is in T_s. Finally, step (A7) introduces null marker occurrences in every column that is not in T_s, unless such a column already features a null marker occurrence.

Algorithm 13 (Armstrong table computation)

Input:

set Σ of standard FDs and standard CCs,

an NFS $nfs(T_s)$ over table schema T such that

$\forall H \in T \exists b \in \mathbb{N}(\Sigma \models_{T_s} card(H) \leq b)$

Output: Armstrong table t for Σ and $nfs(T_s)$

Method: let $c_{H,1}, c_{H,2}, \ldots \in dom(H)$ be distinct

(A0) compute $\forall H \in T(max_{\Sigma[\mathrm{FD}],T_s}(H))$;

(A1) compute $cl\text{-}cl_{\Sigma,T_s}(T)$ as above;

(A2) $t := \emptyset$;

(A3) $i := 1$;

(A4) `for all` $X \in cl\text{-}cl_{\Sigma,T_s}(T)$ where $b_X > 1$ `do`

$t := t \cup \{r_i, \ldots, r_i + b_X - 1\}$ where
$\forall j = i, \ldots, i + b_X - 1$ and $\forall H \in T$

$$r_j(H) := \begin{cases} c_{H,i} & \text{, if } H \in X \\ c_{H,j} & \text{, if } H \in T_s - X \\ \texttt{ni} & \text{, else} \end{cases};$$

$i := i + b_X$;

(A5) `for all` $X \in max^{ess}_{\Sigma,T_s}(T)$ `do`

$Z := \{H \in T - T_s \mid X \in max_{\Sigma,T_s}(H)\}$;
$t := t \cup \{r_i, r_{i+1}\}$ where $\forall H \in T$

$$r_i(H) := \begin{cases} c_{H,i} & \text{, if } H \in XZT_s \\ \texttt{ni} & \text{, else} \end{cases}$$

$$r_{i+1}(H) := \begin{cases} c_{H,i} & \text{, if } H \in X \\ c_{H,i+1} & \text{, if } H \in Z(T_s - X) \\ \texttt{ni} & \text{, else} \end{cases}$$

$i := i + 2$;

(A6) $total(t) := \{H \in T \mid \forall r \in t(r(H) \neq \texttt{ni})\}$;
`if` $total(t) - T_s \neq \emptyset$,
`then return` $t := t \cup \{r_i\}$ where $\forall H \in T$,

$$r_i(H) := \begin{cases} \texttt{ni} & \text{, if } H \in total(t) - T_s \\ c_{H,i} & \text{, else} \end{cases};$$

`else return` t;
`endif`;

Algorithm 13 works correctly.

Theorem 14 *For every input $(T, \Sigma, nfs(T_s))$, where Σ is a set of standard FDs and standard CCs, and $nfs(T_s)$ is an NFS over table schema T such that for all $H \in T$ there is some $b \in \mathbb{N}$ such that $\Sigma \models_{T_s} card(H) \leq b$, Algorithm 13 computes an Armstrong table for Σ and $nfs(T_s)$.*

Theorem 15 *Let Σ be a set of standard FDs and CCs, and $nfs(T_s)$ an NFS over table schema T. Then there is a table over T that is Armstrong for Σ and $nfs(T_s)$ if and only if for all $H \in T$ there is some $b \in \mathbb{N}$ such that $\Sigma \models_{T_s} card(H) \leq b$.*

Example 21 *On input* Schedule, Schedule$_s$ *and Σ from Example 18, Algorithm 13 would compute the table t from Example 17.*

6.5 Complexity considerations

Theorem 16 *Let Σ be a set of standard FDs and CCs, and $nfs(T_s)$ an NFS over table schema T. It can be decided in time $\mathcal{O}(|T|^2 \times ||\Sigma||)$ whether there is an Armstrong table for Σ and $nfs(T_s)$.*

Now we will analyze how well Algorithm 13 does in terms of how well one could potentially do in general. First of all, the problem of computing an Armstrong table for a given set Σ of standard FDs and standard CCs and an NFS over some table schema is precisely exponential in the size of Σ.

Proposition 5 *The problem of computing an Armstrong table for a given set Σ of standard CCs and an NFS* $\mathrm{nfs}(T_s)$ *over table schema T is precisely exponential in the size of Σ.*

It can be shown that Algorithm 13 is quite conservative in its use of time and space, despite the problem of computing Armstrong tables is computationally hard.

Theorem 17 *Algorithm 13 computes an Armstrong table for Σ and $nfs(T_s)$ whose number of rows is at most quadratic in the number of rows of a minimum-sized Armstrong table for Σ and $nfs(T_s)$ and the cardinality of Σ.*

Finally, we show that, in general, there is no most concise way of representing the information inherent in a set of standard CCs and a null-free subschema.

Theorem 18 *Let $\mathcal{C}$ denote the class of FDs and CCs. There is some table schema T, some set Σ of CCs and some NFS $nfs(T_s)$ over T such that Σ has size $\mathcal{O}(n)$, and the size of a minimum-sized $\mathcal{C}$-Armstrong table for Σ and $nfs(T_s)$ is $\mathcal{O}(2^n)$. There is some table schema T, some set Σ of CCs and some NFS T_s over T such that there is a $\mathcal{C}$-Armstrong table for Σ and $nfs(T_s)$ where the number of rows is in $\mathcal{O}(n)$, and the optimal cover of Σ with respect to $nfs(T_s)$ has size $\mathcal{O}(2^n)$.*

Let $T = H_1 \cdots H_{2n}$, $T_s = T$ and let Σ consist of the following standard CCs: for all $i = 1, \ldots, n$, $card(H_{2i-1}H_{2i}) \leq 1$, and for all $i = 1, \ldots, 2n$, $card(H_i) \leq 2$. Then $cl\text{-}cl_{\Sigma,T_s}(T)$ contains the 2^n sets $X \subseteq T$ where for each $i = 1, \ldots, n$ either $H_{2i-1} \in X$ or $H_{2i} \in X$. According to Theorem 12 every Armstrong table for Σ and $nfs(T_s)$ contains a number of rows that is exponential in $||\Sigma||$. This shows the first statement.

Let $T = H_1 H'_1 \cdots H_n H'_n$, $T_s = T$, and let Σ consist of the following standard CCs: for all $i = 1, \ldots, n$, $card(H_i) \leq 3$ and $card(H'_i) \leq 3$, and for all $X = X_1 \cdots X_n$ where $X_i \in \{H_i, H'_i\}$, $card(X) \leq 2$. Then Σ is its own optimal cover, i.e. there is no equivalent set Σ' of standard FDs and standard CCs such that $||\Sigma'|| < ||\Sigma||$. The size $||\Sigma||$ is in $\mathcal{O}(2^n)$. Furthermore, $cl\text{-}cl_{\Sigma,T_s}(T)$ consists of the n sets $T - H_i H'_i$ for $i = 1, \ldots, n$, and the set T, and $max_{\Sigma,T_s}(T)$ consists of the $2n$ sets $T - H_i$ and $T - H'_i$ for $i = 1, \ldots, n$. Thus, Algorithm 13 computes an Armstrong table for Σ and $nfs(T_s)$ whose number of rows is in $\mathcal{O}(n)$.

For these reasons we recommend the use of both representations. Indeed, the representation in form of constraint sets enables design teams to identify constraints that they currently incorrectly perceive as semantically meaningful; and the representation in form of an Armstrong table enables design teams to identify constraints that they currently incorrectly perceive as semantically meaningless.

7 Conclusion and future work

The article has argued that the concept of *Armstrong databases* can provide the foundation for an agile approach to database design that captures well the structure and semantics of the application domain. Specifically, Armstrong databases allow the design team to validate and consolidate their understanding of the underlying application domain, and to communicate this understanding to other stake-holders of the database. We have provided evidence that Armstrong databases for several popular classes of integrity constraints have good structural and computational properties. These properties have been established for classes of integrity constraints over SQL tables, which accommodate partial and duplicate information. Finally, some areas of future work on this subject are outlined.

7.1 Further classes of constraints

The classes of constraints studied in this article are all uni-tabular, i.e. defined over a single table. In practice, there are several other classes of constraints, both uni- and multi-tabular. Further uni-tabular classes include multivalued and full hierarchical dependencies (Fagin 1977, Hartmann & Link 2010*b*); and multi-tabular constraints include referential constraints such as foreign keys and inclusion dependencies. The existence of Armstrong relations has been analyzed for all of these classes (Beeri et al. 1977, Fagin & Vardi 1983, Levene & Loizou 1997). However, it would be interesting to investigate the structural and computational properties of Armstrong tables for such classes, too.

7.2 Other approaches to partial information

In this article we have adopted the *no information* interpretation of a single null marker (Atzeni & Morfuni 1986, Lien 1982, Zaniolo 1984). This is the interpretation that is used by SQL since the single null marker in SQL must be able to accommodate non-existing as well as existing but unknown information. In future work different approaches to partial information should be analyzed. These include different interpretations of a single null marker such as Codd's interpretation `unk` as existing but unknown information (Codd 1979). Levene and Loizou have studied weak and strong possible world semantics associated with this interpretation. They have shown, in particular, that the combined class of weak and strong functional dependencies does enjoy Armstrong tables (Levene & Loizou 1998). An investigation of the structural and computational properties of these Armstrong tables was also suggested for future work. Most of the results presented in our article carry over to weak versions of UCs, FDs and CCs, with only minor changes required. For UCs and FDs these results have been presented in (Hartmann, Ferrarotti, Le & Link 2011). Other approaches to deal with incomplete information comprise or-relations (Imielinski et al. 1995, Libkin & Wong 1996, Vadaparty & Naqvi 1995), fuzzy relations (Sözat & Yazici 2001), rough sets (Ziarko 1991), or world-set decompositions to manage probabilistic information (Antova et al. 2009), among many others.

7.3 Empirical studies

A common perception is that Armstrong relations are useful in the acquisition of data semantics, in particular since errors during the requirements elicitation have the most expensive consequences. Recently, some first empirical evidence was presented for this perception regarding the class of functional dependencies (Langeveldt & Link 2010). For this purpose, the usefulness of Armstrong relations with respect to various measures was investigated. Soundness measures how many of the as semantically meaningful perceived FDs are actually semantically meaningful. Completeness measures how many of the actually semantically meaningful FDs are also perceived as semantically meaningful. Our experiment determines what and how much design teams learn about the application domain in addition to what they know prior to using Armstrong relations. The data analysis suggests that in using Armstrong relations it is not more likely to recognize semantically meaningless FDs which are incorrectly perceived as semantically meaningful, but it is more likely to recognize semantically meaningful FDs that are incorrectly perceived as semantically meaningless. These studies should be extended to other classes of database constraints. The perception is that the more expressive a class of database constraints is, the more design teams can learn from corresponding Armstrong databases. So far, there are not any studies how Armstrong databases can be presented to maximize the discovery of semantically meaningful data dependencies. In the case of FDs for example, is it an advantage to present the Armstrong table as a whole, or to present individually the pairs of rows whose strong agree set are the maximal sets? Is it true that Armstrong relations that only realize the maximal sets induced by an FD set allow us to recognize a higher number of semantically meaningful FDs than Armstrong relations that realize all closed sets induced by the FD set? Similarly for UCs, should we present the Armstrong table as a whole, or should we list individually the first row together with every remaining row (since their strong agree sets are the anti-keys)? It is worthwhile to develop a process model for utilizing Armstrong databases effectively in the database design of a target database. For example, results indicate to utilize Armstrong relations as early as possible in order to prevent design teams from incorrectly perceiving any semantically meaningless FDs as semantically meaningful.

7.4 Informative Armstrong databases

Informative Armstrong databases are used for the *semantic sampling of existing databases* (Bisbal & Grimson 2001, De Marchi & Petit 2007). They constitute small subsets of an existing database that satisfy the same data dependencies. In particular, De Marchi and Petit (De Marchi & Petit 2007) applied the concept of informative Armstrong relations to generate a sample that only used 0.6% of the rows in an existing real-world database and satisfied the exact same set of data dependencies. The small size of this sample enabled data engineers to gain a good understanding about the semantics inherent in a current database instance. The results presented in this article pave the way to develop informative Armstrong tables for existing SQL tables.

7.5 Constraint mining

Closely related to the problem of computing Armstrong databases for a given set Σ of constraints is the problem of computing a cover of the set of constraints satisfied by a given table. The latter problem is known as constraint mining or dependency discovery. The mining of keys and functional dependen-

cies in relations has received considerable attention in the relational model (Huhtala et al. 1999, Mannila & Räihä 1994, Sismanis et al. 2006). The problem has also been studied for referential constraints such as foreign keys (Zhang et al. 2010) and, more generally, inclusion dependencies (De Marchi et al. 2009). Such algorithms should be extended to accommodate database constraints over SQL databases. Ultimately, such algorithms can further support the design methodology introduced in this article. In fact, stakeholders of the database may want to present the database design team with sample data that they think constitutes a typical database instance. The design team may then use the constraint mining algorithms to learn the constraints perceived as semantically meaningful by the stakeholders. Alternatively, the stakeholders may want to change or add data values to the Armstrong databases presented to them. Again, constraint mining algorithms can be applied to the revised database.

References

Abiteboul, S., Hull, R. & Vianu, V. (1995), *Foundations of Databases*, Addison-Wesley.

Akhtar, W., Cortés-Calabuig, A. & Paredaens, J. (2010), Constraints in RDF, *in* 'Proceedings of the 4th International Workshop on Semantics in Data and Knowledge Bases (SDKB)', Vol. 6834 of *Lecture Notes in Computer Science*, Springer, pp. 23–39.

Alexe, B., Kolaitis, P. & Tan, W.-C. (2010), Characterizing schema mappings via data examples, *in* 'Proceedings to the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)', pp. 261–271.

Anthes, G. (2010), 'Happy Birthday, RDBMS!', *Communications of the ACM* **53**(5), 16–17.

Antova, L., Koch, C. & Olteanu, D. (2009), '$10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information', *The VLDB Journal* **18**(5), 1021–1040.

Arenas, M. & Libkin, L. (2004), 'A normal form for XML documents', *ACM Transactions on Database Systems* **29**(1), 195–232.

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

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

Atzeni, P. & Morfuni, N. (1986), 'Functional dependencies and constraints on null values in database relations', *Information and Control* **70**(1), 1–31.

Beeri, C. & Bernstein, P. (1979), 'Computational problems related to the design of normal form relational schemas', *ACM Transactions on Database Systems* **4**(1), 30–59.

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

Beeri, C., Fagin, R. & Howard, J. H. (1977), A complete axiomatization for functional and multivalued dependencies in database relations, *in* 'Proceedings of the International Conference on Management of Data (SIGMOD)', ACM, Toronto, Canada, pp. 47–61.

Bisbal, J. & Grimson, J. (2001), 'Database sampling with functional dependencies', *Information & Software Technology* **43**(10), 607–615.

Boehm, B. (1981), *Software Engineering Economics*, Prentice Hall.

Bojanczyk, M., Muscholl, A., Schwentick, T. & Segoufin, L. (2009), 'Two-variable logic on data trees and XML reasoning', *Journal of the ACM* **56**(3).

Buneman, P., Davidson, S., Fan, W., Hara, C. & W.C., T. (2002), 'Keys for XML', *Computer Networks* **39**(5), 473–487.

CA Technologies (2011), 'ERwin Data Modeler - methods guide', `https://support.ca.com/cadocs/0/e002961e.pdf`, page 86.

Chamberlin, D. & Boyce, R. (1974), SEQUEL: A structured english query language, *in* 'Proceedings of the ACM-SIGMOD Workshop', Vol. 1, ACM, pp. 249–264.

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

Codd, E. F. (1979), 'Extending the database relational model to capture more meaning', *ACM Transactions on Database Systems* **4**(4), 397–434.

Date, C. (2009), *SQL and relational theory*, 1 edn, O'Reilly Media.

De Marchi, F., Lopes, S. & Petit, J.-M. (2009), 'Unary and n-ary inclusion dependency discovery in relational databases', *Journal of Intelligent Information Systems* **32**(1), 53–73.

De Marchi, F., Lopes, S., Petit, J.-M. & Toumani, F. (2003), 'Analysis of existing databases at the logical level: the DBA companion project', *SIGMOD Record* **32**(1), 47–52.

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

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

Demetrovics, J. & Thi, V. (1995*a*), 'Armstrong relations, functional dependencies and strong dependencies', *Computers and Artificial Intelligence* **14**(3).

Demetrovics, J. & Thi, V. (1995*b*), 'Some remarks on generating Armstrong and inferring functional dependencies relation', *Acta Cybernetica* **12**(2), 167–180.

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

Elmasri, R. & Navathe, S. (2010), *Fundamentals of Database Systems*, 6 edn, Addison-Wesley.

Enders, A. & Romback, H. (2003), *A Handbook of Software and Systems Engineering; Empirical Overservations, Laws and Theories*, Addison-Wesley.

Fagin, R. (1977), 'Multivalued dependencies and a new normal form for relational databases', *ACM Transactions on Database Systems* **2**(3), 262–278.

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', *Journal of the ACM* **29**(4), 952–985.

Fagin, R. & Vardi, M. (1983), 'Armstrong databases for functional and inclusion dependencies', *Information Processing Letters* **16**(1), 13–19.

Fagin, R. & Vardi, M. (1986), The theory of data dependencies - a survey, *in* 'Mathematics of Information Processing', American Mathematical Society, pp. 19–72.

Fan, W. & Siméon, J. (2003), 'Integrity constraints for XML', *Journal of Computer and System Sciences* **66**(1), 254–291.

Ferrarotti, F., Hartmann, S. & Link, S. (2011), A precious class of cardinality constraints for flexible XML data processing, *in* 'Proceedings of the 30th International Conference on Conceptual Modeling', Vol. 6998 of *Lecture Notes in Computer Science*, Springer, pp. 175–188.

Gottlob, G. & Libkin, L. (1990), 'Investigation on Armstrong relations, dependency inference, and excluded functional dependencies', *Acta Cybernetica* **9**(4), 385–402.

Hartmann, S. (2001), 'On the implication problem for cardinality constraints and functional dependencies', *Annals of Mathematics and Artificial Intelligence* **33**, 253–307.

Hartmann, S. (2003), Reasoning about participation constraints and Chen's constraints, *in* 'Proceedings of the 14th Australasian Conference on Database Technologies', Vol. 17 of *Conferences in Research and Practice in Information Technology*, ACS, pp. 105–113.

Hartmann, S., Ferrarotti, F., Le, V. & Link, S. (2011), Codd table representations under weak possible world semantics, *in* 'Proceedings of the 22nd International Conference on Database and Expert Systems Applications (DEXA)', number 6860 *in* 'Lecture Notes in Computer Science', Springer, pp. 125–139.

Hartmann, S., Kirchberg, M. & Link, S. (2011), 'Design by example for SQL table definitions with functional dependencies', The VLDB Journal, doi:10.1007/ s00778-011-0239-5.

Hartmann, S., Leck, U. & Link, S. (2011), 'On Codd families of keys over incomplete relations', *The Computer Journal* **54**(7), 1166–1180.

Hartmann, S. & Link, S. (2009), 'Efficient reasoning about a robust XML key fragment', *ACM Transactions on Database Systems* **34**(2).

Hartmann, S. & Link, S. (2010*a*), 'Numerical constraints on XML data', *Information and Computation* **208**(5), 521–544.

Hartmann, S. & Link, S. (2010*b*), When data dependencies over SQL tables meet the Logics of Paradox and S-3, *in* 'Proceedings to the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)', pp. 317–326.

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.

Imielinski, T. & Lipski Jr., W. (1984), 'Incomplete information in relational databases', *Journal of the ACM* **31**(4), 761–791.

Imielinski, T., Van der Meyden, R. & Vadaparty, K. (1995), 'Complexity tailored design: a new design methodology for databases with incomplete information', *Journal of Computer and System Sciences* **51**(3), 405–432.

Katona, G. & Tichler, K. (2006), Some contributions to the minimum representation problem of key systems, *in* 'Proceedings of the 4th International Symposium on Foundations of Information and Knowledge Systems (FoIKS)', Vol. 3861 of *Lecture Notes in Computer Science*, Springer, pp. 240–257.

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

Lausen, G., Meier, M. & Schmidt, M. (2008), SPARQLing constraints for RDF, *in* 'Proceedings of the 11th International Conference on Extending Database Technology (EDBT)', Vol. 261 of *ACM International Conference Proceeding Series*, pp. 499–509.

Lenzerini, M. & Nobili, P. (1990), 'On the satisfiability of dependency constraints in Entity-Relationship schemata', *Information Systems* **15**(4), 453–461.

Levene, M. & Loizou, G. (1997), 'Null inclusion dependencies in relational databases', *Information and Computation* **136**(2), 67–108.

Levene, M. & Loizou, G. (1998), 'Axiomatisation of functional dependencies in incomplete relations', *Theoretical Computer Science* **206**(1-2), 283–300.

Levene, M. & Loizou, G. (1999), *A guided tour of relational databases and beyond*, Springer.

Libkin, L. & Wong, L. (1996), 'Semantic representations and query languages for Or-sets', *Journal of Computer and System Sciences* **52**(1), 125–142.

Liddle, S., Embley, D. & Woodfield, S. (1993), 'Cardinality constraints in semantic data models', *Data & Knowledge Engineering* **11**, 235–270.

Lien, E. (1982), 'On the equivalence of database models', *Journal of the ACM* **29**(2), 333–362.

Maier, D. (1983), *The theory of relational databases*, Pitman Publishing Limited.

Mannila, H. & Räihä, K.-J. (1986), 'Design by example: An application of Armstrong relations', *Journal of Computer and System Sciences* **33**(2), 126–141.

Mannila, H. & Räihä, K.-J. (1992), *Design of Relational Databases*, Addison-Wesley.

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

Martin, J. (1989), *Information Engineering*, Prentice Hall.

Meijer, E. & Bierman, G. (2011), 'A co-relational model of data for large shared data banks', *Communications of the ACM* **54**(4), 49–58.

Rys, M. (2011), 'Scalable SQL', *ACM Queue* **9**(4), 8 pages.

Silva, A. & Melkanoff, M. (1979), A method for helping discover the dependencies of a relation, *in* 'Advances in Data Base Theory', pp. 115–133.

Sismanis, Y., Brown, P., Haas, P. J. & Reinwald, B. (2006), GORDIAN: Efficient and scalable discovery of composite keys, *in* 'Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)', ACM, pp. 691–702.

Sözat, M. & Yazici, A. (2001), 'A complete axiomatization for fuzzy functional and multivalued dependencies in fuzzy database relations', *ACM Fuzzy Sets and Systems* **117**(2), 161–181.

Standish Group (1995), 'Unfinished voyages', The Standish Group International, available on-line at `http://www.standishgroup.com/sample_research/unfinsihed_voyages_1.php`.

Thalheim, B. (1991), *Dependencies in relational databases*, Teubner.

Thalheim, B. (1992), Fundamentals of cardinality constraints, *in* 'Proceedings of the 11th International Conference on the Entity-Relationship Approach', Vol. 645 of *Lecture Notes in Computer Science*, Springer, pp. 7–23.

Thalheim, B. (2000), *Entity-Relationship modeling*, Springer, Heidelberg, Germany.

Ullman, J. D. (1988), *Principles of database and knowledge-base system*, Computer Science Press, Rockville.

Vadaparty, K. & Naqvi, S. (1995), 'Using constraints for efficient query processing in nondeterministic databases', *IEEE Transactions on Knowledge and Data Engineering* **7**(6), 850–864.

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

Zaniolo, C. (1984), 'Database relations with null values', *Journal of Computer and System Sciences* **28**(1), 142–166.

Zhang, M., Hadjieleftheriou, M., Ooi, B., Procopiuc, C. & Srivastava, D. (2010), 'On multi-column foreign key discovery', *PVLDB* **3**(1), 805–814.

Ziarko, W. (1991), The discovery, analysis, and representation of data dependencies in databases, *in* 'Knowledge Discovery in Databases', MIT Press, Cambridge, USA, pp. 195–212.

Zultner, R. (1992), The Deming way: Total quality management for software, *in* 'Proceedings of Total Quality Management for Software', pp. 134–145.