Journal of Universal Computer Science, vol. 15, no. 1 (2009), 195-240
submitted: 11/1/08, accepted: 15/8/08, appeared: 1/1/09 © *J.UCS*

Global Database Design based on Storage Space and Update Time Minimization

Henning Köhler
(University of Queensland, Brisbane, Australia
henning@itee.uq.edu.au)

Abstract: A common approach in designing relational databases is to start with a universal relation schema, which is then decomposed into multiple subschemas. A good choice of subschemas can be determined using integrity constraints defined on the schema, such as functional, multivalued or join dependencies.

In this paper we propose and analyze a new normal form based on the idea of minimizing overall storage space and update costs, and as a consequence redundancy as well. This is in contrast to existing normal forms such as BCNF, 4NF or KCNF, which only characterize the absence of redundancy (and thus space and update time minimality) for a single schema. We show that our new normal form naturally extends existing normal forms to multiple schemas, and provide an algorithm for computing decompositions.

Key Words: database design, universal relation, dependencies, normal forms
Category: H.2.1, F.4

1 Introduction

When designing a database schema, the first question one should ask is "What do I want to achieve?", or in other words: "What characterizes a good design?" This is typically answered by *normal forms*, which aim at characterizing schemas with desirable properties. For operational databases (as opposed to e.g. data warehouses, where the focus is mainly on query optimization) a typical desirable property is the absence of redundancy. If a piece of information is stored multiple times, then this causes problems: The size of our database increases, it is possible to store inconsistent data (although this can often be prevented by enforcing integrity constraints), and updates to this piece of information must be performed multiple times in different places.

However, the problem with trying to prevent redundancy altogether is that it simply isn't always possible.

Example 1. Consider the schema $R = \{Course, Lecturer, Student\}$ with con-

straints $\Sigma = \{Course \rightarrow Lecturer\}$ and instance:

$Course$	$Lecturer$	$Student$
$Algebra$	$Koehler$	123456
$Algebra$	$Koehler$	234567
$Programming$	$Link$	345678
$Programming$	$Link$	456789

Here the lecturer for a course is stored twice, but this redundancy can be removed by decomposing R into $\{Course, Lecturer\}$ and $\{Course, Student\}$:

$Course$	$Lecturer$
$Algebra$	$Koehler$
$Programming$	$Link$

$Course$	$Student$
$Algebra$	123456
$Algebra$	234567
$Programming$	345678
$Programming$	456789

While now the lecturer is stored only once for each course, the set of courses offered is stored twice: once in $\{Course, Lecturer\}$ and again in $\{Course, Student\}$.

The problem of unavoidable redundancy becomes especially obvious if we want to achieve other design goals as well, such as preservation of integrity constraints. We therefore will not aim at eliminating redundancy altogether, but at minimizing the overall amount of redundancy.

This leads to another issue, namely that quantifying the amount of redundancy inherent to a schema is not straight forward. While some attempts at measuring redundancy have been made [Arenas and Libkin, 2003], they turn out to be quite complex, and tailored to specific types of constraints. Instead, we will use the size of instances and time for updates as indirect measures for redundancy, following the intuition that more redundancy leads to a larger size and longer update times. We will argue later that our size model is very useful for (indirectly) measuring redundancy, and show that it is strongly related to update costs, even though it does not truly reflect storage space used in real DBMSs.

Our approach to database normalization is now the following: Among all "suitable" decompositions, select one for which the size of instances (and/or update time) is minimal, to the effect that we also minimize redundancy. Here "suitable" is a generic term, which can be used to capture necessary or desirable design goals, such as e.g. losslessness and preservation of integrity constraints. Different choices for "suitable" lead to different normal forms, but our method for comparing decompositions is independent of that choice (the definition of "suitable" only affects *which* decompositions we need to compare, but not *how* to compare them).

Before we go into more detail, and compare our approach to existing normal forms, we introduce some basic terms from relational database theory.

1.1 Terminology

A *relational database schema* consists of a set of relation schemas. A *relation schema* $R = \{A_1, A_2, \ldots, A_n\}$ is a finite set of *attributes*. Each attribute A_i has a *domain* $dom(A_i)$ associated with it. Domains are arbitrary sets, but unless explicitly stated otherwise, we will assume that domains are countably infinite.

A *relation* r over a relation schema R is a finite or infinite set of tuples, and each element $e_i \in dom(A_i)$ of the tuple corresponds to one attribute $A_i \in R$. Relations over a schema are also commonly referred to as schema instances or tables. Sets of schema instances, one for each relation schema in a database schema, are called database instances.

A vital tool for managing data are *integrity constraints*. They describe, usually in a syntactic manner, what instances of a database schema are acceptable or valid. Formally, an integrity constraint on R is a function mapping relations r on R to $\{true, false\}$. We say that a constraint holds on r if it maps r to $true$.

With each relation schema we associate a set Σ of integrity constraints, in particular *functional dependencies* (FD), *multivalued dependencies* (MVD) and *join dependencies* (JD). These restrict which relations over R we may store. We say that a set Σ of constraints over R *implies* a constraint (or set of constraints) c, written $\Sigma \models c$, if c holds on every relation r over R for which all constraints in Σ hold. If two sets of constraints Σ and Σ' imply each other, we call Σ a *cover* of Σ' (and vice versa). We say that a FD $X \to Y \in \Sigma$ is *redundant* in Σ, if $\Sigma \setminus \{X \to Y\}$ implies $X \to Y$ (and thus is still a cover of Σ).

At this point we need to consider two options: If we allow relations to be infinite, we get a different notion of implication than we get when considering only finite relations. In the latter case, implication is commonly referred to as *finite implication*, and these notions of implications can be different [Casanova et al., 1982]. In this work we shall consider only finite relations. However, implication and finite implication are actually the same for functional and join dependencies [Maier, 1983], so most of our results also hold for infinite relations.

A functional dependency on R is an expression of the form $X \to Y$ (read "X *determines* Y") where X and Y are subsets of R. For attribute sets X, Y and attribute A we will write XY short for $X \cup Y$ and A short for $\{A\}$. We say that a FD $X \to Y$ *holds* on a relation r over R if every pair of tuples in r that coincides on all attributes in X also coincides on all attributes in Y. We call a FD $X \to Y$ *trivial* if $Y \subseteq X$. Trivial FDs are the only FDs which hold on every relation. A set $X \subseteq R$ is a *key* of R w.r.t. a set Σ of integrity constraints on R, if Σ implies $X \to R$. Note that some authors use the term 'key' only for minimal keys, and call keys which may not be minimal 'superkeys'.

To testing whether a FD $X \to Y$ is implied by a set Σ of other FDs, we compute the *closure* X^* of the left hand side X, which is the set of all attributes determined by X:

$$X^* := \{A \in R \mid X \to A \in \Sigma^*\}$$

In the rare case where the set Σ is not clear from the context, we write $X^{*\Sigma}$.

Once computed, we only need to check whether the right hand side Y is a subset of X^*. Computing X^* can be done quickly using the well-known closure algorithm, which can be implemented to run in linear time [Beeri and Bernstein, 1979].

COMMENT.

Algorithm "closure"

INPUT: set of FDs Σ, attribute set X
OUTPUT: X^*, the closure of X w.r.t. Σ

$X^* := X$
while $\exists X' \to Y \in \Sigma$ with $X' \subseteq X^*, Y \not\subseteq X^*$ do
    $X^* := X^*Y$
end

For a set $X \subseteq R$ we denote the *projection* of r onto the attributes in X by $r[X]$. The *join* of two relations $r[X]$ and $r[Y]$ is a relation on $X \cup Y$:

$$r[X] \bowtie r[Y] := \left\{ t \,\middle|\, \begin{array}{l} \exists t_1 \in r[X], t_2 \in r[Y]. \\ t[X] = t_1 \wedge t[Y] = t_2 \end{array} \right\}$$

A join dependency on R is an expression of the form $\bowtie [R_1, \ldots, R_n]$ where the R_i are subsets of R with $\bigcup R_i = R$. We say that the JD $\bowtie [R_1, \ldots, R_n]$ holds on r if the decomposition $\{R_1, \ldots, R_n\}$ is *lossless* for r, i.e., if

$$r[R_1] \bowtie \ldots \bowtie r[R_n] = r$$

A multivalued dependency on R is a join dependency $\bowtie [R_1, R_2]$ with only two subschemas. It is usually written as $X \twoheadrightarrow Y$ where $X = R_1 \cap R_2$ and $Y = R_1 \setminus R_2$ or $Y = R_2 \setminus R_1$.

1.2 Normal Forms

A common approach in designing relational databases is to start with a universal relation schema, which is then decomposed into multiple subschemas. A good choice of subschemas can often be determined using integrity constraints defined on the original schema. Note that we only consider relations which are globally

consistent, i.e., relations for subschemas are obtained as projections of a universal relation.

To ensure that the schemas of a decomposition $\mathcal{D} = \{R_1, \ldots, R_n\}$ with $R_i \subseteq R$ can hold the same data as the original schema R, we must ask for a decomposition that has the *lossless join* property, i.e., the join dependency $\bowtie [R_1, \ldots, R_n]$ must be implied by Σ.

Another common requirement is that the decomposition should be dependency preserving or *faithful*, i.e., the dependencies on the schemas R_i which are implied by Σ should form a cover of Σ. This allows a database management system to check constraints for individual relations only, without having to compute their join. The *projection* of a set Σ of FDs onto $R_i \subseteq R$ is

$$\Sigma[R_i] := \{X \to Y \in \Sigma \mid XY \subseteq R_i\}$$

Thus a decomposition $\mathcal{D} = \{R_1, \ldots, R_n\}$ is dependency preserving if

$$\left(\bigcup \Sigma^*[R_i]\right)^* = \left(\bigcup \Sigma_i\right)^* = \Sigma^*$$

where Σ_i is a cover for $\Sigma^*[R_i]$ (when describing the decomposition, we usually want to represent $\Sigma^*[R_i]$ by a smaller cover for it). Note that it is not sufficient to only project Σ onto the R_i, rather than Σ^*.

Example 2. Let $R = ABC$ with constraints $\Sigma = \{A \to B, B \to C\}$. Then $\Sigma[AC] = \emptyset$, although $A \to C$ is a FD on AC which is implied by Σ.

While it is possible to define dependency preservation for other types of constraints as well (e.g. join dependencies), we do not require this here.

Normal forms are syntactic descriptions of good relation or database schemas. A number of normal forms have been proposed, depending on the types of integrity constraints used. We shall introduce one of them briefly here.

A relation schema R is in *Boyce-Codd Normal Form* w.r.t. a set Σ of FDs on R if and only if for every non-trivial FD $X \to Y \in \Sigma$ the left hand side (LHS) X is a key for R, i.e., $X \to R \in \Sigma^*$, where

$$\Sigma^* := \{X \to Y \mid X, Y \subseteq R, \Sigma \vDash X \to Y\}$$

For a more thorough introduction see e.g. [Levene and Loizou, 1999; Maier, 1983; Mannila and Räihä, 1987].

1.3 Problems with Existing Normal Forms

Many normal forms proposed so far, such as BCNF, 4NF or KCNF, characterize the absence of redundancy [Arenas and Libkin, 2003; Vincent, 1998]. This is desirable for several reasons, foremost the avoidance of update anomalies [Maier,

1983] and minimization of storage space [Biskup, 1995]. However, these normal forms have significant drawbacks. First, they only consider a single relation schema, instead of considering all schemas in a decomposition together. While this is not strictly true for the normal form proposed by Topor and Wang in [Wang and Topor, 2005], their approach only considers pairs of schemas at a time, and thus cannot capture redundancy across larger schema sets.

The common generalization to multiple schemas is that the whole schema collection is in that normal form, if every schema taken individually is. But this means that those normal forms cannot capture redundancy which exists across multiple relations. As a trivial example, we can duplicate a schema. Clearly the extra schema is then superfluous in the whole schema collection, but each schema taken individually may still be redundancy free. But even if no schemas or attributes in a schema collection are superfluous, the design may not be desirable.

Example 3. Let $R = ABCD$ and $\Sigma = \{AB \to CD, CD \to B\}$. Then R is not in BCNF, but has a dependency preserving BCNF decomposition into the subschemas ABC, ABD, BCD.

However, for any instance r of R, the projections of r onto the schemas ABC and ABD together already take up more space than the original relation r: no tuples are lost in the projection since AB is a key, and the attributes A and B are stored twice:

A	B	C	D
1	**1**	1	1
2	**1**	1	1
1	2	1	2

$\Rightarrow$

A	B	C
1	**1**	1
2	**1**	1
1	2	1

A	B	D
1	**1**	1
2	**1**	1
1	2	2

B	C	D
1	1	1
2	1	2

While we have not defined what redundancy means for multiple schemas, it seems intuitively clear that this decomposition should not be called "redundancy free". From a storage space point-of-view, it is clearly less desirable than the original schema R. Also updates may take longer in decomposed form: If we want to e.g. update the value for B which is determined by $C = 1, D = 1$, we need to change only 2 tuples in r, but a total of 5 tuples in the decomposed representation.

The second big problem is that dependency preserving decompositions into these normal forms do not always exist [Beeri and Bernstein, 1979]. Thus, when faced with such a case, a designer must either accept the loss of some dependencies, or cannot achieve the normal form in question. We believe that what a normal form should do, is the following:

> Characterize "good" representations (i.e., decompositions) of a schema, in such a way that a "good" representation does always exist. Furthermore, the definition of "good" should have a clear semantic motivation.

In the following we will propose a normal form which meets this criterion. In section 2 we define our *Domination Normal Form (DNF)* based on different ordering pairs for comparing decompositions. These ordering pairs will then be shown to be equivalent in section 3, leading to a syntactical characterization for DNF. This is then used in section 4 to compare DNF to existing normal forms. Finally, an algorithm for computing decompositions is given in section 5.

A short version of this paper appeared in [Köhler, 2007].

2 Minimization as Normal Form

The approach we suggest is the following: among a set of suitable decompositions (e.g. the set of all lossless, or lossless and dependency preserving decompositions), we characterize the "best" ones. We do so by defining an order on the decompositions, such that the "best" decompositions are the minimal ones with respect to that order.

This leaves the question of when to call one decomposition better than another one. The motivation for many normal forms proposed so far has been the elimination of redundancy (and with it, the absence of update-anomalies). This may suggest to define a quantitative measure of redundancy over multiple schemas, similar to the work of Arenas and Libkin in [Arenas and Libkin, 2003].

We take a different approach here: instead of trying to minimize redundancy, we try to minimize the size of instances and time for updates. Intuitively this should lead to similar results, an assumption which is supported by the findings of Biskup in [Biskup, 1995], but measures for size and update time appear easier to construct than measures for redundancy (cf. [Arenas and Libkin, 2003]). In the following we will define and motivate different orders on decompositions. By proving them to be equivalent, we will establish a syntactic characterization for a semantically motivated definition.

2.1 Ordering by Size of Instances

Our first approach measures the space required to store an instance. For that we need to know for each element of a domain how much storage space it requires. We represent this knowledge by associating with each domain Dom a size function

$$size : Dom \to \mathbb{N}$$

Comment. Consider e.g. the following domains:

- $STRING$ containing strings of arbitrary length
- $STRING[40]$ containing strings of length up to 40

- INT containing arbitrarily large integers
- $INT(64)$ containing all 64-bit integers
- $BOOLEAN$ containing the values $TRUE$ and $FALSE$

A realistic measure for the size of a string might be its length, the size of an integer i might be defined as $log(i)$, and the size of $TRUE$ and $FALSE$ might be one.

In this work we shall assume that all domains are infinite, and that the size functions on them are positive and unbounded, i.e., can grow arbitrarily large. This can be justified as follows: While not all domains are truly infinite, they often contain far more elements than the number of subschemas in a typical decomposition (e.g. 256^{40} for $STRING[40]$ or 2^{64} for $INT(64)$). Treating these domains as infinite will allow us to draw a sharp boundary between small (bounded) increases in size from duplicated attributes on one hand, and potentially large (unbounded) increases in size from instances with large numbers of tuples on the other. We note that this argument fails for domains such as $BOOLEAN$, but in this work we will not concern ourselves with such cases.

However, for most domains occurring in practice, a constant size function would be more realistic, e.g. 40 for $STRING[40]$ or 8 for $INT(64)$. The problem with constant size functions is that the total size of instances under different decompositions hardly ever varies by more than a small constant factor, which makes it hard to distinguish "good" decompositions from "bad" ones (at least our approach will not work).

On the other hand, by allowing arbitrarily large attribute values, we can capture the fact that an attribute value may be stored arbitrarily many times, which may cause update problems. When assuming unbounded domains, the problems of minimizing storage space and update time become equivalent.

As it will turn out, the assumptions about infinite domains and unbounded size functions are all we need to characterize our new normal form, i.e., we do not require detailed knowledge about the actual size functions.

Definition 1. For a relation r over R and a decomposition $\mathcal{D} = \{R_1, \ldots, R_n\}$ of $R = \bigcup R_j$ we denote the decomposition of r by $\mathcal{D}$ as

$$r[\mathcal{D}] := \{r[R_1], \ldots, r[R_n]\}$$

where $r[R_j]$ is the projection of r onto the attributes in R_j. When talking about the tuples in $r[\mathcal{D}]$ containing an attribute A, we will mean the tuples from relations $R_j \in \mathcal{D}$ with $A \in R_j$.

Definition 2 (Size). Let $R = \{A_1, \ldots, A_k\}$ be a schema. For a finite relation

r over R we define the size of r as

$$size(r) := \sum_{t \in r} \sum_{i=1}^{k} size(\pi_{A_i}(t))$$

where $\pi_{A_i}(t)$ denotes the projection of tuple t onto the attribute A_i. We then define the size of the decomposition of r by $\mathcal{D} = \{R_1, \ldots, R_n\}$ of R as

$$size(r[\mathcal{D}]) := \sum_{j=1}^{n} size(r[R_j])$$

While this gives us a suitable definition of size for any instance, we wish to compare decompositions w.r.t. the size of all valid instances. If for every valid instance r on R a decomposition $\mathcal{D}_1$ requires no more storage space than a decomposition $\mathcal{D}_2$, then $\mathcal{D}_1 \leq \mathcal{D}_2$ should certainly hold, indicating that $\mathcal{D}_1$ is "at most as big" and thus "at least as good" as $\mathcal{D}_2$. Recall that we only consider suitable decompositions, e.g. lossless or lossless and dependency preserving ones.

This alone, however, is not sufficient to characterize good decompositions: for an instance r containing only a single element, the trivial decomposition $\{R\}$ requires less storage space than any other lossless decomposition, as those typically need to duplicate some attributes. It would be hard to argue though that decomposition is never necessary. So how can we distinguish decompositions finer, based on the size of instances?

Example 4. Let $R = ABC$ and $\Sigma = \{B \to C\}$. R can be faithfully decomposed into $\mathcal{D} = \{AB, BC\}$. Clearly every relation r decomposed by $\mathcal{D}$ (which is a set of relations) is at most twice as large as r. On the other hand, for every natural number k we can construct a relation

$r =$

A	B	C
1	1	"a very long string"
2	1	"a very long string"
⋮	⋮	⋮
$k+1$	1	"a very long string"

which is more than k times larger than in decomposed form:

$r[AB] =$

A	B
1	1
2	1
⋮	⋮
$k+1$	1

$r[BC] =$

B	C
1	"a very long string"

This observation motivates the following definitions, which compare decompositions similarly to the "big-O" comparison (e.g. $3x^2 + x \in O(x^2)$) from complexity theory.

Definition 3 (Size (c-)domination). Let R be a schema with constraints Σ and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. We say that $\mathcal{D}_1$ *c-dominates* $\mathcal{D}_2$ (where "c" stands for complexity) if there exists a constant k such that for all finite relations r over R that satisfy Σ we have

$$size(r[\mathcal{D}_1]) \leq k \cdot size(r[\mathcal{D}_2])$$

We further say that $\mathcal{D}_1$ *dominates* $\mathcal{D}_2$ if the above relationship holds for $k = 1$. We abbreviate c-domination and domination as $\mathcal{D}_1 \leq_c \mathcal{D}_2$ and $\mathcal{D}_1 \leq \mathcal{D}_2$, respectively. We say that $\mathcal{D}_1$ *strictly (c-)dominates* $\mathcal{D}_2$, written $\mathcal{D}_1 <_{(c)} \mathcal{D}_2$, if $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ but not vice-versa.

It is easy to see that both domination and c-domination are reflexive and transitive, and thus are pre-orders. Clearly domination implies c-domination.

Proposition 4. *Let $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1$ dominates $\mathcal{D}_2$ then $\mathcal{D}_1$ c-dominates $\mathcal{D}_2$.*

Note however that strict domination does *not* imply strict c-domination. In example 3 the original schema $ABCD$ strictly dominates the decomposition $\{ABC, ABD, BCD\}$, but both decompositions are equivalent w.r.t. c-domination. Sometimes both criteria, domination and c-domination, are used to characterize the best decomposition for a schema, as the following example shows.

Example 5. Let $R = ABCDE$ and $\Sigma = \{AB \rightarrow CD, B \rightarrow E\}$. Then the decomposition $\mathcal{D}_1 = \{ABC, ABD, BE\}$ is minimal w.r.t. c-domination but strictly dominated by $\mathcal{D}_2 = \{ABCD, BE\}$. The trivial decomposition $\{R\}$ is minimal w.r.t. domination but strictly c-dominated by both $\mathcal{D}_1$ and $\mathcal{D}_2$.

It is however not trivial to verify that all these statements hold. In section 2.4 we will introduce a syntactical characterization of domination and c-domination, which makes this task easier. For now, consider the following sample relation:

$r =$

A	B	C	D	E
1	1	1	1	"very long string"
2	1	1	2	"very long string"
3	1	2	1	"very long string"

Here we get the projected relations:

$r[\mathcal{D}_1] =$

A	B	C
1	1	1
2	1	1
3	1	2

$+$

A	B	D
1	1	1
2	1	2
3	1	1

$+$

B	E
1	"very long string"

$r[\mathcal{D}_2] =$

A	B	C	D
1	1	1	1
2	1	1	2
3	1	2	1

$+$

B	E
1	"very long string"

Values for attributes C, D and E occur equally often in $\mathcal{D}_1$ and $\mathcal{D}_2$, since ABC, ABD and $ABCD$ are all key schemas, and thus no tuples are lost in the projection. Values for attributes A and B are stored twice in $\mathcal{D}_1$ though, and only once in $\mathcal{D}_2$. Thus $\mathcal{D}_1$ and $\mathcal{D}_2$ are equivalent w.r.t. c-domination (differing by a factor of at most two), but $\mathcal{D}_2$ strictly dominates $\mathcal{D}_1$. Furthermore, $\{R\}$ is not dominated by either $\mathcal{D}_1$ nor $\mathcal{D}_2$, since the latter decompositions duplicate the attribute B. It is strictly c-dominated by both of them though, since values for attribute E can appear arbitrarily more often in R than in BE.

We are now ready to define our normal form based on the idea of minimizing storage space.

Definition 5 (Domination Normal Form). Let R be a schema with constraints Σ and $\mathcal{D}$ be a decomposition of R. We say that $\mathcal{D}$ is in *domination normal form* (DNF) if $\mathcal{D}$ is minimal w.r.t. both domination and c-domination, with minimal meaning that no strictly smaller decomposition exists among a given set of 'suitable' decompositions.

Note that this definition depends on the choice of which decompositions we consider 'suitable'. We will investigate two different cases (though other choices might be of interest as well): the set of all lossless, and the set of all lossless and dependency preserving decompositions. In each case, we effectively obtain a different DNF.

As the number of suitable decompositions of a given schema is finite, there must exist a decomposition among them which is minimal w.r.t. domination, as well as a (possibly different) schema which is minimal w.r.t. c-domination. It is however not clear yet whether a decomposition into DNF always exists, i.e., one which is minimal w.r.t. both criteria at once. We will show this next.

Theorem 6. *Every schema has a decomposition into DNF.*

Proof. We use the fact that domination implies c-domination. The c-domination pre-order induces a partition of all the (suitable) decompositions of R into equivalence classes and defines a partial order on these equivalence classes. Let EQ be a minimal equivalence class w.r.t. that order. Choose $\mathcal{D}$ to be minimal w.r.t. domination among the decompositions in EQ. We claim that $\mathcal{D}$ is minimal w.r.t. domination among *all* decompositions of R, and thus in DNF. Let $\mathcal{D}'$ be any decomposition with $\mathcal{D}' \leq \mathcal{D}$. Then $\mathcal{D}' \leq_c \mathcal{D}$, and since $\mathcal{D}$ is minimal w.r.t. c-domination, $\mathcal{D}' \in EQ$. But $\mathcal{D}$ is also minimal w.r.t. domination in EQ, and thus $\mathcal{D} \leq \mathcal{D}'$. Thus no decomposition $\mathcal{D}'$ strictly dominates $\mathcal{D}$.

2.2 Ordering by Update Time

While storage space (size) is rather straight-forward to measure (the only real issue arising is that of unbounded domains), update time can be measured in more than one reasonable way. Also it is not so clear what exactly constitutes an update. In the following we will introduce a very simple notion of updates (single value updates) and update time, and based on this another pair of domination and c-domination orders. This will prove equivalent to the orders by size, which in particular illustrates why the assumption of unbounded domains is useful, albeit not realistic.

Definition 7 (Update Location). Let R be a schema, A an attribute in R and φ a boolean condition which can be interpreted for any tuple over R. Furthermore let r be an instance of R.

We call the pair (φ, A) an *update location* for R. It updates A for the tuples

$$r|_\varphi := \{t \in r \mid \varphi(t)\}$$

For a value $a \in dom(A)$, the updated relation is then

$$r[A \mapsto a \mid \varphi] := \{t' \mid t'[A] = a \wedge \exists t \in r|_\varphi . t'[R \setminus A] = t[R \setminus A]\} \cup r|_{\neg\varphi}$$

We do not want to allow arbitrary kinds of updates though. We restrict ourselves to updating a single value (a single value a_{old} is replaces by a new value a in multiple tuples), and the updated relation should still be valid, at least for most new values a. By valid we mean satisfying all constraints specified on R.

Definition 8 (Valid Update Location). Let r be a valid instance of R. We say that the update location (φ, A) is *valid* for r, if

(i) $t[A] = t'[A]$ for all $t, t' \in r|_\varphi$

(ii) for every value $a \in dom(A)$ which does not appear in r, the updated relation $r[A \mapsto a \mid \varphi]$ is a valid instance of R

We can now measure the time it takes to update a relation under different decompositions. Since this is independent from the new value a assigned to A, we only require the update location. Basically, we measure the number of "cells" in the relation tables which need to be updated.

Definition 9 (Count). For a finite relation r over schema R and an attribute A we define the *count* of A on r as

$$count_A(r) := \begin{cases} |r| & \text{if } A \in R \\ 0 & \text{if } A \notin R \end{cases}$$

where $|r|$ denotes the number of tuples in r. We then define the count of A on r decomposed by a decomposition $\mathcal{D} = \{R_1, \ldots, R_n\}$ of R as

$$count_A(r[\mathcal{D}]) := \sum_{j=1}^{n} count_A(r[R_j])$$

Definition 10 (Update Time). Let r be a valid instance of R, and (φ, A) a valid update location for r. Furthermore let $\mathcal{D}$ be a decomposition of r. Then the *update time* for relation r at location (φ, A) while decomposed under $\mathcal{D}$ is

$$time(r, \varphi, A, \mathcal{D}) := count_A(r|_\varphi[\mathcal{D}])$$

Example 6. Consider again $R = ABCD$ and $\Sigma = \{AB \to CD, CD \to B\}$ from Example 3, with decomposition $\mathcal{D} = \{ABC, ABD, BCD\}$ and instance r:

A	B	C	D
1	**1**	1	1
2	**1**	1	1
1	2	1	2

The update location (φ, B) with $\varphi(a, b, c, d) :=$ "$c = 1 \wedge d = 1$" is valid for r. When decomposed, $r|_\varphi[\mathcal{D}]$ takes the form

A	B	C
1	**1**	1
2	**1**	1

A	B	D
1	**1**	1
2	**1**	1

B	C	D
1	1	1

which gives us an update time of $time(r, \varphi, B, \mathcal{D}) = 2 + 2 + 1 = 5$.

While this notion of update time is rather simplistic, is suffices for our purposes. We can now describe domination and c-domination w.r.t. update time.

Definition 11 (Update Time (c-)domination). Let R be a schema with decompositions $\mathcal{D}_1, \mathcal{D}_2$. We say that $\mathcal{D}_1$ *c-dominates* $\mathcal{D}_2$ w.r.t. update time, if there

exists a constant k such that for all valid relations r over R and update locations (φ, A) which are valid for r, we have

$$time(r, \varphi, A, \mathcal{D}_1) \leq k \cdot time(r, \varphi, A, \mathcal{D}_2)$$

Again, we say that $\mathcal{D}_1$ *dominates* $\mathcal{D}_2$ w.r.t. update time, if the above relationship holds for $k = 1$, and use the same notation for this as before.

2.3 Ordering by Attribute Count of Instances

We now introduce an order pair which is similar to both order pairs introduced so far. Instead of measuring the total size of instances, we count the number of tuples an attribute appears in. This gives us domination and c-domination pre-orders for each attribute, and we can then combine these to get another pair of orderings for decompositions.

While this ordering lacks the intuitive motivation of the previous orderings, it will be a useful tool for connecting them with the syntactic ordering presented in the next section.

Definition 12 (Single Attribute (c-)domination). Let R be a schema with FDs Σ, A an attribute and $\mathcal{D}_1, \mathcal{D}_2$ decompositions of R. We say that $\mathcal{D}_1$ *c-dominates* $\mathcal{D}_2$ w.r.t. A if there exists a constant k such that for all finite relations r over R that satisfy Σ we have

$$count_A(r[\mathcal{D}_1]) \leq k \cdot count_A(r[\mathcal{D}_2])$$

We further say that $\mathcal{D}_1$ *dominates* $\mathcal{D}_2$ w.r.t. A if the above holds for $k = 1$.

Thus, for each attribute A, we get a c-domination and domination pre-orders. We combine those pre-orders by intersection.

Definition 13 (Attribute Count (c-)domination). Let R be a schema and $\mathcal{D}_1, \mathcal{D}_2$ decompositions of R. We say that

$$\mathcal{D}_1 \left\{ \begin{array}{l} \text{dominates} \\ \text{c-dominates} \end{array} \right\} \mathcal{D}_2 \text{ w.r.t. attribute count}$$

if for every attribute $A \in R$ we have $\mathcal{D}_1 \left\{ \begin{array}{l} \text{dominates} \\ \text{c-dominates} \end{array} \right\} \mathcal{D}_2$ w.r.t. A.

2.4 Ordering by Containing Schema Closures

The previous order pairs introduced are defined by considering all valid instances. This is not very practical if we wish to actually decide for two given decompositions whether one (c-)dominates the other. We therefore present a further pair

of orders, which is defined by considering only the decompositions, rather than instances on them.

The approach we use is similar to attribute counting. The count of an attribute depends on the set of schemas it lies in. While it also depends on the instance r, it is easy to show that the number of tuples in $r[R_j]$ is determined by the closure R_j^* of R_j (and by r). Recall that the closure R_j^* of R_j under Σ is

$$R_j^* := \{A \in R \mid \Sigma \vDash R_j \to A\}$$

where Σ is a given set of constraints on R. In this work, we shall mainly be interested in functional, multi-valued and join dependencies.

Lemma 14. *Let R be a schema with arbitrary constraints Σ, and $X \subseteq R$. Then for all relations r on R we have $|r[X]| = |r[X^*]|$.*

Proof. We can obtain $r[X]$ by projecting from $r[X^*]$. The only way for the number of tuples to decrease, is for $r[X^*]$ to contain different tuples which are identical on X. But this cannot happen since X functionally determines X^*.

This motivates the following definitions.

Definition 15 (Containing Schema Closures). Let R be a schema with constraints Σ, and $\mathcal{D}$ a decomposition of R. Then for any attribute $A \in R$ we define the *containing schema closures* (CSC) of A in $\mathcal{D}$ as the *multiset*

$$CSC_A(\mathcal{D}) = \{R_j^* \mid A \in R_j \in \mathcal{D}\}$$

The idea behind this definition is that the containing schema closure of an attribute A represents the attribute count for A in a way which does not rely on particular instances, but still allows comparison of different decompositions. It is necessary to use multisets rather than sets to represent the correct attribute count.

Example 7. Consider again the schema $R = ABCDE$ with constraints $\Sigma = \{AB \to CD, B \to E\}$ from example 5, and the decompositions

$$\mathcal{D}_1 = \{ABC, ABD, BE\}$$
$$\mathcal{D}_2 = \{ABCD, BE\}$$

They produce the multisets

$$CSC_A(\mathcal{D}_1) = \{ABCDE, ABCDE\}$$
$$CSC_A(\mathcal{D}_2) = \{ABCDE\}$$

which indicate that, for any relation r on R, the attribute A appears in twice as many tuples in $r[\mathcal{D}_1]$ as in $r[\mathcal{D}_2]$. Using sets would hide this difference.

We will now compare decompositions using the respective CSCs of all attributes. For that we need mappings between multi-sets. We allow different instances of the same value in the source domain to map to different values, and call a mapping injective if a value in the target domain is mapped to at most as often as it occurs in the target domain.

Example 8. Given the multisets $M_1 = \{1, 1, 2\}$ and $M_2 = \{a, a, b\}$, consider the following multiset "mappings" from M_1 to M_2:

$$f_1 = \{1 \mapsto a, 1 \mapsto a, 2 \mapsto b\}$$
$$f_2 = \{1 \mapsto a, 1 \mapsto b, 2 \mapsto a\}$$
$$f_3 = \{1 \mapsto b, 1 \mapsto b, 2 \mapsto a\}$$
$$f_4 = \{1 \mapsto a, 2 \mapsto a, 2 \mapsto b\}$$

Mappings f_1, f_2 are injective, while mapping f_3 is not. The multiset f_4 does not constitute a mapping from M_1 to M_2, since it maps the value 2 twice.

Definition 16 (Inclusion Domination). Let M_1, M_2 be two multisets (or sets for part (i)) of attribute sets. We say that

(i) M_1 *weakly inclusion-dominates* M_2 if there exists a mapping $f : M_1 \to M_2$ with $e \subseteq f(e)$ for all $e \in M_1$.

(ii) M_1 *strongly inclusion-dominates* M_2 if there exists an injective mapping $f : M_1 \to M_2$ with $e \subseteq f(e)$ for all $e \in M_1$.

Definition 17 (CSC-Domination). Let $\mathcal{D}_1, \mathcal{D}_2$ be two decompositions of R. We say that $\mathcal{D}_1$ *weakly/ strongly csc-dominates* $\mathcal{D}_2$ if for all attributes $A \in R$ we have that $CSC_A(\mathcal{D}_1)$ weakly/strongly inclusion-dominates $CSC_A(\mathcal{D}_2)$.

Example 9. Consider again $R = ABCDE$ with $\Sigma = \{AB \to CD, B \to E\}$, and

$$\mathcal{D}_1 = \{ABC, ABD, BE\}$$
$$\mathcal{D}_2 = \{ABCD, BE\}$$

They produce the containing schema closures

$$CSC_A(\mathcal{D}_1) = \{ABCDE, ABCDE\}$$
$$CSC_A(\mathcal{D}_2) = \{ABCDE\}$$
$$CSC_B(\mathcal{D}_1) = \{ABCDE, ABCDE, BE\}$$
$$CSC_B(\mathcal{D}_2) = \{ABCDE, BE\}$$
$$CSC_C(\mathcal{D}_1) = CSC_C(\mathcal{D}_2) = \{ABCDE\}$$
$$CSC_D(\mathcal{D}_1) = CSC_D(\mathcal{D}_2) = \{ABCDE\}$$
$$CSC_E(\mathcal{D}_1) = CSC_E(\mathcal{D}_2) = \{BE\}$$

Looking at attribute A, we can see that $CSC_A(\mathcal{D}_1)$ weakly inclusion dominates $CSC_A(\mathcal{D}_2)$, while $CSC_A(\mathcal{D}_2)$ strongly inclusion dominates $CSC_A(\mathcal{D}_1)$. Similarly for attribute B, and for attributes C, D, E we get strong inclusion domination in both directions. Thus $\mathcal{D}_1$ weakly csc-dominates $\mathcal{D}_2$, while $\mathcal{D}_2$ strongly (and also weakly) csc-dominates $\mathcal{D}_1$.

We will prove in the next section that weak csc-domination implies c-domination, and that strong csc-domination implies domination. While the opposite does not hold for arbitrary types of constraints, we will be able to show that it holds for sets of functional dependencies, and in the case of weak csc-domination/ c-domination also for multi-valued and join dependencies. Thus we obtain a syntactic characterization for c-domination and, at least in the case of functional dependencies, for domination. For multi-valued dependencies, domination does not imply strong csc-domination, as will become evident in example 16. Finding a syntactic characterization for domination in this case is an open problem.

3 Equivalence of Orderings

We will show that, if the only integrity constraints on R are functional dependencies, then all order pairs defined in section 2 are identical. If multi-valued and join dependencies are also allowed, then domination w.r.t. size, update time or attribute count need not imply weak csc-domination, but we will show that the other equivalences between the orders in question still hold.

As multi-valued dependencies are just a special case of join dependencies, it suffices to consider only functional and join dependencies. Unless indicated otherwise, we will assume throughout this section that functional and join dependencies are the only types of integrity constraints occurring.

Note that equivalence of (c-)domination w.r.t. size, update time and attribute count can be established for a larger classes of integrity constraints (those which permit permutations of domain elements for individual attributes, as well as constant selection) just as easily. However, when constructing sample relations to show equivalence of (c-)domination and (weak/strong) csc-domination, we have to consider individual types of integrity constraints. Here we only have results for functional and join dependencies.

The remainder of section 3 is devoted to proving the following theorems:

Theorem 18. *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. Then $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. size iff $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. attribute count.*

Theorem 19. *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. Then $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. update time iff $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. attribute count.*

Theorem 20. *Let R be a schema with constraints Σ (arbitrary, not just FDs and JDs), and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1 \left\{ \begin{array}{l} \text{weakly} \\ \text{strongly} \end{array} \right\}$ csc-dominates $\mathcal{D}_2$ then $\mathcal{D}_1 \left\{ \begin{array}{l} \text{c-dominates} \\ \text{dominates} \end{array} \right\} \mathcal{D}_2$.*

Theorem 21. *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1$ c-dominates $\mathcal{D}_2$, then $\mathcal{D}_1$ weakly csc-dominates $\mathcal{D}_2$.*

Theorem 22. *Let R be a schema with functional dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1$ dominates $\mathcal{D}_2$, then $\mathcal{D}_1$ strongly csc-dominates $\mathcal{D}_2$.*

Note that these equivalence results are independent of which decompositions we consider "suitable": even if a decomposition is deemed unsuitable, e.g. because it is not lossless or not dependency preserving, we can still compare it with any other decomposition. The empty decomposition is the smallest decomposition w.r.t. any of the orders, but clearly it should never be considered suitable.

3.1 Size vs. Attribute Count

We start by showing that the orders defined by size and attribute count are identical. Recall that we assume that all domains are infinite, and that the size functions associated with them are positive and unbounded.

Lemma 23. *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. size, then $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. attribute count.*

Proof. If $\mathcal{D}_1$ does not c-dominate $\mathcal{D}_2$ w.r.t. attribute count, then for every integer k there exists a relation r and an attribute A with

$$count_A(r[\mathcal{D}_1]) > k \cdot count_A(r[\mathcal{D}_2])$$

For each k and associated r and A, we will construct a relation r' for which

$$size(r'[\mathcal{D}_1]) > k \cdot size(r'[\mathcal{D}_2])$$

holds. This shows that $\mathcal{D}_1$ does not c-dominate $\mathcal{D}_2$ w.r.t. size. For domination we only need to consider the case $k = 1$.

The construction works as follows. Since the relations in $r[\mathcal{D}_1]$ with attribute A contain more than k times as many tuples as those in $r[\mathcal{D}_2]$, there must be an attribute value v_A for A which appears more than k times as often in $r[\mathcal{D}_1]$ as in $r[\mathcal{D}_2]$. We construct r' from r by substituting every occurrence of v_A by

a new value v'_A which does not appear in r. As Σ contains only functional and join dependencies, these constraints still hold for r'. Let o_1, o_2 be the number of occurrences of v_A in $r[\mathcal{D}_1], r[\mathcal{D}_2]$. We choose v'_A sufficiently large, i.e., such that

$$size(v'_A) > \frac{k \cdot size(r[\mathcal{D}_2]) - size(r[\mathcal{D}_1])}{o_1 - k \cdot o_2} + size(v_A)$$

This gives us (note that $o_1 - k \cdot o_2 > 0$):

$$\begin{aligned}
(o_1 - k \cdot o_2) \cdot (size(v'_A) - size(v_A)) &> k \cdot size(r[\mathcal{D}_2]) - size(r[\mathcal{D}_1]) \\
size(r[\mathcal{D}_1]) + o_1 \cdot (size(v'_A) - size(v_A)) &> k \cdot size(r[\mathcal{D}_2]) + \\
&\quad k \cdot o_2 \cdot (size(v'_A) - size(v_A)) \\
size(r'[\mathcal{D}_1]) &> k \cdot size(r'[\mathcal{D}_2])
\end{aligned}$$

This concludes the proof.

Lemma 24. *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. attribute count, then $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. size.*

Proof. For every k, r ($k = 1$ for domination) with

$$size(r[\mathcal{D}_1]) > k \cdot size(r[\mathcal{D}_2])$$

we need to construct a relation r' such that for some attribute A we get

$$count_A(r'[\mathcal{D}_1]) > k \cdot count_A(r'[\mathcal{D}_2])$$

Here we use that the attribute values occurring in $r[\mathcal{D}_1]$ and $r[\mathcal{D}_2]$ are the same. Since $size(r[\mathcal{D}_1]) > k \cdot size(r[\mathcal{D}_2])$, there must exist some attribute value v_A of an attribute A which occurs more than k times as often in $r[\mathcal{D}_1]$ than in $r[\mathcal{D}_2]$. We construct r' from r by selecting exactly those tuples which have the value v_A on attribute A. As Σ contains only functional and join dependencies, these constraints still hold for r'. And clearly we now have $count_A(r'[\mathcal{D}_1]) > k \cdot count_A(r'[\mathcal{D}_2])$.

We can combine the last two lemmas.

Theorem 25 (18). *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. Then $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. size iff $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. attribute count.*

3.2 Update Time vs. Attribute Count

We next compare (c-)domination w.r.t. update time and attribute count.

Lemma 26. *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. update time, then $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. attribute count.*

Proof. If $\mathcal{D}_1$ does not (c-)dominate $\mathcal{D}_2$ w.r.t. attribute count, then for every integer k (or just for $k = 1$) there exists an attribute A and a relation r_k with

$$count_A(r_k[\mathcal{D}_1]) > k \cdot count_A(r_k[\mathcal{D}_2])$$

Thus there must exist a value $a_{old} \in r_k[A]$ which also appears that much more often in $r_k[\mathcal{D}_1]$ than in $r_k[\mathcal{D}_2]$. Denoting the set of tuples t in r_k with $t[A] = a_{old}$ as $r_k|_\varphi$ with $\varphi =$ "$A = a_{old}$", we get

$$count_A(r_k|_\varphi[\mathcal{D}_1]) > k \cdot count_A(r_k|_\varphi[\mathcal{D}_2]) \tag{1}$$

Since Σ contains only functional and join dependencies, both of which allow permutations of domain elements for individual attributes, the updated relation $r_k[A \mapsto a|\varphi]$ is valid for all values $a \in dom(A)$ not occurring in r_k. Thus (φ, A) is a valid update location for r_k, and equation (1) shows that $\mathcal{D}_1$ does not (c-)dominate $\mathcal{D}_2$ w.r.t. update time.

Lemma 27. *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. attribute count, then $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. update time.*

Proof. If $\mathcal{D}_1$ does not (c-)dominate $\mathcal{D}_2$ w.r.t. update time, then for every integer k (or just for $k = 1$) there exists a relation r_k and an update location (φ, A) valid for r_k such that

$$count_A(r_k|_\varphi[\mathcal{D}_1]) > k \cdot count_A(r_k|_\varphi[\mathcal{D}_2]) \tag{2}$$

holds. Since R contains only a finite number of attributes, we may assume A to be fixed for all k (although φ may differ).

Now let $a \in dom(A)$ be a new value for A not occurring in r_k. As (φ, A) is valid for r_k, the updated relation $r_k[A \mapsto a|\varphi]$ is valid. Given that Σ contains only functional and join dependencies, both of which permit constant selection, the subrelation $r'_k := r_k|_\varphi[A \mapsto a]$ is also valid. Since all tuples in $r_k|_\varphi$ have the same value on A (by definition of valid update location), no tuples are lost due to duplication when updating. Thus equation (2) gives us

$$count_A(r'_k[\mathcal{D}_1]) > k \cdot count_A(r'_k[\mathcal{D}_2])$$

which shows that $\mathcal{D}_1$ does not (c-)dominate $\mathcal{D}_2$ w.r.t. A and thus not w.r.t. attribute count, as witnessed by the relations r'_k.

Again we combine those lemmas into one theorem.

Theorem 28 (19). *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. Then $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. update time iff $\mathcal{D}_1$ (c-)dominates $\mathcal{D}_2$ w.r.t. attribute count.*

3.3 Attribute Count vs. Containing Schema Closures - Part I

We will now show that the orders defined by attribute count and containing schema closures are actually the same. One direction of implication is easy.

Theorem 29 (20). *Let R be a schema with constraints Σ (arbitrary, not just FDs and JDs), and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1 \left\{ \begin{matrix} \textit{weakly} \\ \textit{strongly} \end{matrix} \right\}$ csc-dominates $\mathcal{D}_2$ then $\mathcal{D}_1 \left\{ \begin{matrix} \textit{c-dominates} \\ \textit{dominates} \end{matrix} \right\} \mathcal{D}_2$.*

Proof. Let r be any relation on R and A some attribute in R.

If $\mathcal{D}_1$ weakly csc-dominates $\mathcal{D}_2$, then for every schema $R_1 \in \mathcal{D}_1$ with $A \in R_1$ there exists a schema $R_2 \in \mathcal{D}_2$ with $A \in R_2$ and $R_1^* \subseteq R_2^*$. By Lemma 14 we have $|R_1| \leq |R_2|$, and each such schema R_2 is mapped to at most $|\mathcal{D}_1|$ times. Therefore the number of tuples containing attribute A in $r[\mathcal{D}_1]$ is at most $|\mathcal{D}_1|$ times larger than the number of tuples with A in $r[\mathcal{D}_2]$. Thus $\mathcal{D}_1$ c-dominates $\mathcal{D}_2$ with $k = |\mathcal{D}_1|$.

If $\mathcal{D}_1$ strongly csc-dominates $\mathcal{D}_2$, then by Lemma 14 and due to the injectivity of the mapping f in Definition 16, the number of tuples with attribute A in $r[\mathcal{D}_1]$ is no larger than the number of those in $r[\mathcal{D}_2]$. Thus $\mathcal{D}_1$ dominates $\mathcal{D}_2$.

To show implication in the other direction, we will assume that $\mathcal{D}_1$ does not weakly or strongly csc-dominate $\mathcal{D}_2$, and construct example relations which show that $\mathcal{D}_1$ does not c-dominate or dominate $\mathcal{D}_2$. These constructions will require some work, and we devote the next subsection to them.

3.4 Subset Construction

Our goal is to construct relations over a schema $\mathcal{D}$ with functional and join dependencies Σ, for which the number of tuples in their projection onto non-key subschemas varies by an arbitrarily large factor.

Definition 30 (k-Reducing). Let R be a schema with constraints Σ and $\mathcal{D} = \{R_1, \ldots, R_n\}$ a decomposition of R. We say that a non-empty relation r over R is *k-reducing* w.r.t. $\mathcal{D}$ for an integer k if

(i) all constraints in Σ hold on r, and

(ii) for every subschema $R_j \in \mathcal{D}$ which is not a key of R, the projection of r onto R_j contains at most $\frac{1}{k}$ times as many tuples as r.

Example 10. Consider the schema $R = ABCD$ with constraint set

$$\Sigma = \{AB \to C, C \to AB, A \to D, B \to D\}$$

and the (faithful and lossless) decomposition $\mathcal{D} = \{ABC, AD, BD\}$. We can construct a 2-reducing instance of R w.r.t. $\mathcal{D}$ as follows:

$r =$

A	B	C	D
1	1	(1, 1)	1
1	2	(1, 2)	1
2	1	(2, 1)	1
2	2	(2, 2)	1

The FDs in Σ clearly hold, and the projections of r onto AD and BD contain only 2 tuples, compared to 4 tuples in r.

We constructed the example relation above by creating two variables v_A, v_B with domain $\{1, 2\}$, and for each value pair $(v_A = a, v_B = b)$ creating a tuple in r, making the value of A dependent on a, the value of B dependent on b, the value of C dependent on both and the value of D dependent on neither. We formalize this idea as follows.

Definition 31 (Lifting). Let Ω be a finite set and R be a set of attributes, where each $A_i \in R$ has a subset S_i of Ω associated with it. For a positive integer k we define the *k-mappings* of Ω as the total functions from Ω into $\{1, \ldots, k\}$. For each k-mapping f we define its *lifting* onto R as the tuple t_f on R, in which the attribute A_i has as value the partial function

$$f|_{S_i} : \Omega \to \{1, \ldots, k\} := \{x \mapsto y \in f \mid x \in S_i\}$$

We get the *k-lifting* of R by taking all k-mappings of Ω and lifting them all onto R. Note that the k-lifting is a set of tuples on R, and thus a relation on R.

While the attribute values constructed by lifting k-mappings are partial functions rather than elements of the attribute's domain, it is easy to see that one could always substitute those values with values from the proper domains (recall that we assumed domains to be infinite) to get an isomorphic relation on R. Note that while this substitution may affect the size of relations, it has no affect on attribute count. As we are only trying to relate the containing schema closure measures to attribute count, rather than size measures directly, we will not worry about size or domains any further.

When giving examples, we use attributes $A, B, \ldots$ rather than $A_1, A_2, \ldots$, and denote their associated subsets by $S_A, S_B, \ldots$ instead of $S_1, S_2, \ldots$.

Example 11. The given definitions can be related to example 10 as follows. We have

$$\Omega = \{v_A, v_B\}, S_A = \{v_A\}, S_B = \{v_B\}, S_C = \{v_A, v_B\}, S_D = \emptyset$$

Writing the total function $\{v_A \mapsto a, v_B \mapsto b\}$ short as (a, b), we obtain the set of all 2-mappings as

$$\{1,2\}^{\Omega} = \left\{ \begin{matrix} (1,1), \\ (1,2), \\ (2,1), \\ (2,2) \end{matrix} \right\}.$$

These 2-mappings get lifted onto R as follows:

	A	B	C	D
$(1,1) \curvearrowright$	$(1,-)$	$(-,1)$	$(1,1)$	$(-,-)$
$(1,2) \curvearrowright$	$(1,-)$	$(-,2)$	$(1,2)$	$(-,-)$
$(2,1) \curvearrowright$	$(2,-)$	$(-,1)$	$(2,1)$	$(-,-)$
$(2,2) \curvearrowright$	$(2,-)$	$(-,2)$	$(2,2)$	$(-,-)$

This is isomorphic to relation r from Example 10.

The lifting of a k-mapping depends on the sets S_i associated with the attributes A_i, and these sets S_i are the only free choices we have in our construction. Note that elements of Ω which do not appear in any S_i do not affect the construction, thus we may as well assume that $\Omega = \bigcup S_i$. Given a schema R with constraints Σ and a decomposition $\mathcal{D}$ of R, we want to choose the sets S_i in such a way that the constructed relation is k-reducing. We will first consider the case where Σ contains only FDs.

Definition 32 (Associated Set/Depending Attributes). For a subschema $X \subseteq R$ we call the set $S_X := \bigcup_{A_i \in X} S_i$ the subset *associated* with X. For $v \in \Omega$ we call $R_v := \{A_i \in R \mid v \in S_i\}$ the set of attributes *depending* on v.

Lemma 33. *Let Ω and R be as in Definition 31, X a subschema of R and S_X its associated subset of cardinality s. Let r_k be the k-lifting of R and x_k the k-lifting of X. Then*

$$x_k = r_k[X]$$

and x_k contains exactly k^s tuples.

Proof. Let f be a k-mapping of Ω, and $t_{f,R}$ and $t_{f,X}$ its liftings onto R and X, respectively. By definition $t_{f,X} = t_{f,R}[X]$, and since this holds for all f we get $x_k = r_k[X]$.

Clearly there are k^s k-mappings of S_X. We show that x_k contains k^s tuples by giving a one-to-one mapping between tuples of x_k and k-mappings of S_X.

The components of a tuple $t_{f,X} \in x_k$ are the partial functions $f|_{S_i}$. By taking their union we obtain $f|_{S_X}$, which is a k-mapping of S_X. Conversely, we can obtain $t_{f,X}$ from $f|_{S_X}$ by restricting $f|_{S_X}$ to the associated sets $S_i \subseteq S_X$ with $A_i \in X$. This gives us the one-to-one mapping we wanted. COMMENT.

By taking the union of the components (i.e., attribute values) of $t_{f,X}$, which are all partial functions $\Omega \to \{1, \ldots, k\}$, we obtain $f|_{S_X}$. On the other hand, all the components of $t_{f,X}$ are restrictions of $f|_{S_X}$, which gives us a one-to-one correspondence between tuples of x_k and k-mappings of S_X. There are k^s k-mappings of S_X.

Lemma 34. *Let Ω and R be as in Definition 31, $k \geq 2$ and r_k the k-lifting of R. Let further $X, Y \subseteq R$, and S_X, S_Y be their associated subsets. Then the FD $X \to Y$ holds on r_k iff $S_X \supseteq S_Y$.*

Proof. (1) Let $S_X \supseteq S_Y$, and let $t_1, t_2 \in r_k$ be tuples in r_k. If $t_1[X] = t_2[X]$ then the k-mappings f_1, f_2 which were lifted onto R to obtain $t_1 = t_{f_1}$ and $t_2 = t_{f_2}$ have the same restriction to S_X, that is $f_1|_{S_X} = f_2|_{S_X}$. Since $S_X \supseteq S_Y$ this implies $f_1|_{S_Y} = f_2|_{S_Y}$, and therefore $t_1[Y] = t_2[Y]$. Thus $X \to Y$ holds on r_k.

(2) Let $S_X \not\supseteq S_Y$, i.e., there exists some $v \in S_Y \setminus S_X$. Let f_1, f_2 be k-mappings of Ω which differ only on v. Then for their liftings $t_{f_1}, t_{f_2} \in r_k$ onto R we have $t_{f_1}[X] = t_{f_2}[X]$ but $t_{f_1}[Y] \neq t_{f_2}[Y]$. Thus $X \to Y$ does not hold on r_k.

Definition 35 (Open/Closed). Let R be a schema with constraints Σ. We say that a subschema $X \subseteq R$ is *open* if its complement $R \setminus X$ is *closed* under Σ, that is,

$$(R \setminus X)^* = R \setminus X$$

Lemma 36. *Let Ω, R, Σ and S_i be as in Definition 31, $k \geq 2$ and r_k the k-lifting of R. Let Σ contain only FDs. Then Σ holds on r_k iff for every $v \in \Omega$ the subschema R_v of attributes depending on v (Definition 32) is open.*

Proof. (1) Let all R_v be open. Assume that some FD $X \to Y \in \Sigma$ does not hold on r_k. Then by Lemma 34 there exists some $v \in S_Y \setminus S_X$. This means that X contains no attributes which depend on v, i.e., no attributes in R_v, so $X \subseteq R \setminus R_v$. Since $R \setminus R_v$ is closed under Σ this implies $Y \subseteq R \setminus R_v$, and thus $v \notin S_Y$. This contradicts $v \in S_Y \setminus S_X$ and disproves our assumption.

(2) Let R_v not be open for some $v \in \Omega$. Then there exists an attribute $A \in R_v$ with $\Sigma \vDash R \setminus R_v \to A$. Due to $A \in R_v$ we have $v \in S_A$, and clearly $v \notin S_{R \setminus R_v}$ by Definition 32. Thus the FD $R \setminus R_v \to A$ does not hold on r_k by Lemma 34.

Lemmas 33 and 36 indicate how we should construct the S_i to obtain a relation on R which is k-reducing w.r.t. some decomposition $\mathcal{D}$. For every subschema $R_j \in \mathcal{D}$ which is not a key of R, there should be an element $v \in \Omega$ which does

not lie in the subset S_{R_j} associated with R_j. At the same time, the set R_v should be open. This leads to the following construction.

Subset Construction for FDs: Let R be a schema with FDs Σ, and $\mathcal{D}$ a decomposition of R. For every subschema $R_j \in \mathcal{D}$ we form its closure R_j^*, and add a unique element v_j to every set S_i for which $A_i \notin R_j^*$. The set Ω from Definition 31 is then

$$\Omega := \bigcup S_i = \{v_j \mid R_j \in \mathcal{D} \text{ and } R_j^* \neq R\}$$

Note that if $\mathcal{D}$ contains multiple subschemas with the same closure, it would suffice to consider only one of them when constructing the sets S_i.

Example 12. Consider again the schema R and decomposition $\mathcal{D}$ from examples 10 and 11. Applying the subset construction we get

$$\begin{array}{lll} ABC^* = ABCD & \curvearrowright \text{do nothing} \\ BD^* \;\; = BD & \curvearrowright \text{add } v_{BD} \text{ to } S_A, S_C \\ AD^* \;\; = AD & \curvearrowright \text{add } v_{AD} \text{ to } S_B, S_C \end{array}$$

This gives us the associated subsets

$$S_A = \{v_{BD}\}, S_B = \{v_{AD}\}, S_C = \{v_{BD}, v_{AD}\}, S_D = \{\}$$

which (except for element names) are the same as in example 11.

Theorem 37. *Let R be a schema with FDs Σ, and $\mathcal{D}$ a decomposition of R. Let the S_i be constructed using the subset construction for FDs. Then for every k the k-lifting r_k of R is k-reducing w.r.t. $\mathcal{D}$.*

Proof. (*i*) Let $v = v_j$ be added to Ω when considering R_j during the subset construction. $R_v = R \setminus R_j^*$ is open, so Σ holds on r_k by Lemma 36.

(*ii*) For every subschema $R_j \in \mathcal{D}$ which is not a key of R, $S_R = \Omega$ contains at least one more element than S_{R_j}, namely the element v_j which was associated with all attributes in $R \setminus R_j^*$. Thus by Lemma 33, the projection of r_k onto R_j contains at most $\frac{1}{k}$ times as many tuples as r_k.

We will now generalize the subset construction to work with functional and join dependencies. We first need to establish when a join dependency holds on a k-lifting. For that, we need some basic terminology from hypergraph theory.

Definition 38 (Hypergraph). A *hypergraph* H on a vertex set V is a set of subsets of V, i.e., $H \subseteq \mathcal{P}(V)$. The elements of H are called *edges*.

Definition 39 (Connected). A hypergraph H is *disconnected* if V can be partitioned into two disjoint non-empty sets $V = V_1 \cup V_2$ such that every edge $e \in H$ lies completely in V_1 or V_2, i.e., $e \subseteq V_1$ or $e \subseteq V_2$. Otherwise we call H *connected*.

Definition 40 (Synchronization Hypergraph). Let Ω and R be as in Definition 31 and $\bowtie [R_1, \ldots, R_n]$ a join-dependency on R. For every $v \in \Omega$ the *synchronization hypergraph* of v w.r.t. $\bowtie [R_1, \ldots, R_n]$ is the projection of the hypergraph $\{R_1, \ldots, R_n\}$ onto R_v (the set of attributes depending on v), i.e.,

$$sync(v) = \{R_1 \cap R_v, \ldots, R_n \cap R_v\}$$

Lemma 41. *Let Ω and R be as in Definition 31, $k \geq 2$ and r_k the k-lifting of R. Let further $\bowtie [R_1, \ldots R_n]$ be a join-dependency on R. Then $\bowtie [R_1, \ldots, R_n]$ holds on r_k iff for all $v \in \Omega$ the synchronization hypergraph H_v of v w.r.t. $\bowtie [R_1, \ldots, R_n]$ is connected.*

Proof. Let S_R be the subset of Ω associated with R. COMMENT. Since elements in $\Omega \setminus S_R$ do not influence r_k, and their synchronization hypergraphs are empty and thus connected, we may as well assume that $\Omega = S_R$. By definition, the join dependency $\bowtie [R_1, \ldots R_n]$ holds on r_k iff

$$r_k[R_1] \bowtie \ldots \bowtie r_k[R_n] =: j_k \subseteq r_k$$

(1) Consider a single tuple $t \in j_k$, and recall that all its attribute values are partial functions $\Omega \to \{1, \ldots, k\}$. Let us denote the union of these partial functions by $\bigsqcup t$, which is a relation $f_t \subseteq \Omega \times \{1, \ldots, k\}$. If f_t is a function, then t is the lifting of f_t onto R, and thus $t \in r_k$. On the other hand, every $t' \in r_k$ is the lifting of some function $f : \Omega \to \{1, \ldots, k\}$, and thus $\bigsqcup t' = f$. Together this gives us that a tuple $t \in j_k$ lies in r_k iff $\bigsqcup t$ is a function.

(2) Let $t \in j_k, v \in \Omega$. For every subschema R_j from $\bowtie [R_1, \ldots, R_n]$ we have that $t[R_j] = t'[R_j]$ for some $t' \in r_k$, and therefore that $\bigsqcup t[R_j]$ is a partial function. Furthermore we have

$$\bigsqcup t[R_i \cup R_j] = \bigsqcup t[R_i] \cup \bigsqcup t[R_j]$$

If there exists an attribute $A \in R_i \cap R_j$, then every v associated with A has the same image (one could say that the mapping of v is "synchronized", hence the name "synchronization hypergraph") under $\bigsqcup t[R_i]$ as it has under $\bigsqcup t[R_j]$, so $\bigsqcup t[R_i \cup R_j]$ is a function for v, i.e., it maps v to only a single value. For a given $v \in \Omega$ let R_v be the set of attributes depending on v. Then such an attribute A exists for v iff $R_i \cap R_v$ and $R_j \cap R_v$ are not disjoint, i.e., iff the partial synchronization hypergraph with edges $R_i \cap R_v$ and $R_j \cap R_v$ is connected.

(3) If H_v is connected, we can use the argument of (2) multiple times to show that for any $t \in j_k$ $\bigsqcup t$ is a function for v. If all H_v are connected, $\bigsqcup t$ is a function, which by (1) shows $t \in r_k$, and thus $j_k \subseteq r_k$. This proves the "if" direction of the lemma.

(4) If H_v is disconnected for some v (which implies $v \in S_R$), then we can partition $\{R_1, \ldots, R_n\}$ into two sets P_1, P_2 such that the attribute sets $\bigcup P_1 \cap R_v$

and $\bigcup P_2 \cap R_v$ are disjoint and non-empty. Since $k \geq 2$ we can find two functions $f_1, f_2 : \Omega \to \{1, \ldots, k\}$ which differ exactly for v. Let $t_1, t_2 \in r_k$ be their liftings onto R. Then by definition of j_k there exists a tuple $t \in j_k$ with

$$t[P_1] = t_1[P_1] \text{ and } t[P_2] = t_2[P_2]$$

But this gives us

$$\bigsqcup t = \bigsqcup t_1[P_1] \cup \bigsqcup t_2[P_2]$$

which is not a function since $\bigsqcup t_1[P_1]$ and $\bigsqcup t_2[P_2]$ map v to different values. As $\bigsqcup t$ is not a function we get $t \notin r_k$ by (1), and thus $j_k \not\subseteq r_k$. This shows the "only if" direction and completes the proof.

Definition 42 (Dependency Basis). Let $X \subseteq R$ be an attribute set and Σ a set of functional and join dependencies on R. The *dependency basis* of X w.r.t. Σ (and R) is the finest partition $DB_\Sigma(X)$ of $R \setminus X^*$ into non-empty sets, such that for every $Y \in DB_\Sigma(X)$ the multivalued dependency $X \twoheadrightarrow Y$ is implied by Σ. Where Σ is clear from the context we will just write $DB(X)$ for $DB_\Sigma(X)$.

It is well known that such a unique finest partition always exists [Mannila and Räihä, 1987]. Note that some texts define the dependency basis of X as the finest partition of R rather than $R \setminus X^*$. As Σ implies $X \twoheadrightarrow A$ for all $A \in X^*$, this partitions X^* into sets each consisting of only a single attribute. Thus knowing the partition of $R \setminus X^*$ immediately gives us the partition of R as well. We chose the given definition as it makes some formulations easier.

COMMENT. The following inference rules for functional and multivalued dependencies are well-known to be correct [Maier, 1983; Mannila and Räihä, 1987], and will be useful in what follows:

$$\frac{X \twoheadrightarrow Y}{X \twoheadrightarrow XY} \text{ (Augmentation)}$$

$$\frac{X \twoheadrightarrow Y \quad Y \twoheadrightarrow Z}{X \twoheadrightarrow Z \setminus Y} \text{ (Pseudo-Transitivity)}$$

$$\frac{X \twoheadrightarrow Y \quad Y \to Z}{X \to Z \setminus Y} \text{ (Mixed Pseudo-Transitivity)}$$

Lemma 43. *Let Σ be a set of functional and join dependencies on R. Then for every $X \subseteq R$, every $Y \in DB(X)$ is open, i.e., $R \setminus Y$ is closed under Σ.*

Proof. Assume that $R \setminus Y$ were not closed under Σ. Then for some attribute $A \in Y$ we have $\Sigma \vDash R \setminus Y \to A$. Since $Y \in DB(X)$ we have $\Sigma \vDash X \twoheadrightarrow R \setminus Y$. Using these two facts together we can derive

$$\frac{X \twoheadrightarrow R \setminus Y \quad R \setminus Y \to A}{X \to A} \quad \text{(Mixed Pseudo-Transitivity)}$$

Thus $A \in X^* \cap Y$, which is a contradiction to $Y \in DB(X)$, since $DB(X)$ partitions $R \setminus X^*$.

Our intermediate goal is to construct (for given R, Σ and $\mathcal{D}$) the sets S_i, such that for every k the resulting k-lifting r_k is k-reducing. We have found such a construction for the case where Σ contains only functional dependencies. When Σ contains join dependencies as well, we need to adapt our construction, since otherwise the join dependencies in Σ need not hold on r_k.

Example 13. Consider the schema $R = ABCD$ with constraints

$$\begin{aligned}\Sigma &= \{\bowtie [AB, AC, AD]\} \\ &\equiv \{A \twoheadrightarrow B|C|D\}\end{aligned}$$

and decomposition $\mathcal{D} = \{AB, AC, AD\}$. Using the subset construction for functional dependencies, we would get the sets

$$S_A = \{\}, S_B = \{v_{AC}, v_{AD}\}, S_C = \{v_{AB}, v_{AD}\}, S_D = \{v_{AB}, v_{AC}\}$$

which in turn lead to the 2-lifting (again writing partial functions as tuples):

$r_2 =$

A	B	C	D
$(-,-,-)$	$(-,1,1)$	$(1,-,1)$	$(1,1,-)$
$(-,-,-)$	$(-,1,2)$	$(1,-,2)$	$(1,1,-)$
$(-,-,-)$	$(-,2,1)$	$(1,-,1)$	$(1,2,-)$
$(-,-,-)$	$(-,2,2)$	$(1,-,2)$	$(1,2,-)$
$(-,-,-)$	$(-,1,1)$	$(2,-,1)$	$(2,1,-)$
$(-,-,-)$	$(-,1,2)$	$(2,-,2)$	$(2,1,-)$
$(-,-,-)$	$(-,2,1)$	$(2,-,1)$	$(2,2,-)$
$(-,-,-)$	$(-,2,2)$	$(2,-,2)$	$(2,2,-)$

It is easy to check that $\bowtie [AB, AC, AD]$ does not hold on r_2.

We observe that the join dependency $\bowtie [AB, AC, AD]$ in the example above does not hold because the sets S_B, S_C, S_D share common elements, i.e., they are not pairwise disjoint. This establishes a connection between the values of B, C and D for every tuple in r_k. To avoid such connections, we could associate

v_{AB}, v_{AC} and v_{AD} only with the attributes of one set Y in $DB(AB), DB(AC)$ and $DB(AD)$, respectively.

While in general it is critical that for every $v \in \Omega$ the set R_v of attributes associated with v is open, in order to ensure that the FDs in Σ hold on r_k (Lemma 36), Lemma 43 ensures us that every $Y \in DB(X)$ is open, for any $X \subseteq R$. This motivates the following construction:

Generalized Subset Construction: Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}$ a decomposition of R. For every subschema $R_j \in \mathcal{D}$ with $R_j^* \neq R$ we form its dependency basis $DB(R_j)$. Then for some (arbitrary) set $Y \in DB(R_j)$, we add a unique element v_j to every set S_i for which $A_i \in Y$.

Note that when Σ contains only functional dependencies, we have

$$DB(R_j) = \{R \setminus R_j^*\}$$

Thus the generalized subset construction is identical to the subset construction for FDs in such cases, which justifies its name.

Example 14. Consider again the schema $R = ABCD$ with constraints

$$\Sigma = \{\bowtie [AB, AC, AD]\}$$

and decomposition $\mathcal{D} = \{AB, AC, AD\}$. Using the generalized subset construction we might get the sets (depending on the choices for Y)

$$S_A = \{\}, S_B = \{v_{AC}\}, S_C = \{v_{AD}\}, S_D = \{v_{AB}\}$$

which lead to the 2-lifting:

$r_2 =$

A	B	C	D
$(-,-,-)$	$(-,1,-)$	$(-,-,1)$	$(1,-,-)$
$(-,-,-)$	$(-,1,-)$	$(-,-,2)$	$(1,-,-)$
$(-,-,-)$	$(-,2,-)$	$(-,-,1)$	$(1,-,-)$
$(-,-,-)$	$(-,2,-)$	$(-,-,2)$	$(1,-,-)$
$(-,-,-)$	$(-,1,-)$	$(-,-,1)$	$(2,-,-)$
$(-,-,-)$	$(-,1,-)$	$(-,-,2)$	$(2,-,-)$
$(-,-,-)$	$(-,2,-)$	$(-,-,1)$	$(2,-,-)$
$(-,-,-)$	$(-,2,-)$	$(-,-,2)$	$(2,-,-)$

It is easy to verify that $\bowtie [AB, AC, AD]$ now holds on r_2.

Comment.

Example 15. Let $R = ABCDE$ with the dependencies

$$\Sigma = \{A \to B, A \twoheadrightarrow C\}$$

and the decomposition $\mathcal{D} = \{AB, ACDE\}$. Applying the generalized subset construction we get

$$\begin{aligned} DB(AB) &= \{C, DE\} \curvearrowright \text{add } v_1 \text{ to } S_C \text{ and } v_2 \text{ to } S_D, S_E \\ DB(ACDE) &= \emptyset \qquad \curvearrowright \text{do nothing} \end{aligned}$$

which gives us the associated subsets

$$S_A = \emptyset, S_B = \emptyset, S_C = \{v_1\}, S_D = \{v_2\}, S_E = \{v_2\}$$

Using them to construct the 2-lifting of R we obtain the relation

$r =$

A	B	C	D	E
1	1	1	1	1
1	1	1	2	2
1	1	2	1	1
1	1	2	2	2

Proposition 44. *For every join dependency $\bowtie [R_1, R_2, \ldots]$ we have:*

$$\bowtie [R_1, R_2, \ldots] \vDash \bowtie [R_1 \cup R_2, \ldots]$$

i.e., if we replace any two (or more) subschemas in a join dependency by their union, we obtain an implied join dependency.

Proof. Clear by definition of join dependency.

Theorem 45. *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}$ a decomposition of R. Let the S_i be constructed using the generalized subset construction. Then the k-lifting r_k of R is k-reducing, for every $k \in \mathbb{N}$.*

Proof. Part (ii) of the k-reducing property can be shown as in Theorem 37. The difficulty lies in showing part (i), namely that all constraints in Σ hold. This is clear for functional dependencies by lemmas 43 and 36.

Assume that some join dependency $\bowtie [R_1, \ldots, R_n] \in \Sigma$ does not hold for r_k, $k \geq 2$ ($k = 1$ is trivial). Then by Lemma 41 there must be some $v \in \Omega$ for which the synchronization hypergraph

$$H_v = \{R_1 \cap R_v, \ldots, R_n \cap R_v\}$$

of v w.r.t. $\bowtie [R_1, \ldots, R_n]$ is disconnected. Then we can partition R_v, the set of attributes depending on v, into non-empty sets

$$R_v = H_1 \cup H_2$$

such that every R_j from $\bowtie [R_1, \ldots, R_n]$ is disjoint to H_1 or H_2. Let U_1 be the union of R_j disjoint to H_2, and U_2 of those disjoint to H_1. Then by Proposition 44 we have:

$$\Sigma \vDash \bowtie [R_1, \ldots, R_n] \vDash \bowtie [U_1, U_2]$$

Now let X be the closure of the subschema in $\mathcal{D}$ for which v was added. By construction we have $R_v \in DB(X)$, so that we can partition R into X, R_v and the remaining attributes $Z := R \setminus (X \cup R_v)$, so that

$$\begin{aligned} R &= X \cup Z \cup R_v \\ &= X \cup Z \cup H_1 \cup H_2 \end{aligned}$$

Then $U_1 \subseteq X \cup Z \cup H_1$ and $U_2 \subseteq X \cup Z \cup H_2$, and thus

$$\begin{aligned} \Sigma &\vDash \bowtie [U_1, U_2] \\ &\vDash \bowtie [X \cup Z \cup H_1, X \cup Z \cup H_2] \\ &\equiv XZ \twoheadrightarrow H_1 \end{aligned}$$

Since Z is the union of elements of the dependency basis of X, we have $X \twoheadrightarrow Z$, and thus

$$\frac{X \twoheadrightarrow Z \quad XZ \twoheadrightarrow H_1}{X \twoheadrightarrow H_1} \quad \text{(Pseudo-Transitivity)}$$

This is a contradiction, since $\emptyset \neq H_1 \subsetneq R_v$, and $R_v \in DB(X)$.

3.5 Attribute Count vs Containing Schema Closures - Part II

We are now ready to complete the equivalence proof for attribute count and csc-domination orders.

Lemma 46. *Let Σ be a set of functional and join dependencies on R, and $X, Y \subseteq R$. Then $\Sigma \cup \{Y \to R\} \vDash X \to Y$ iff $\Sigma \vDash X \to Y$.*

Proof. If Σ implies $X \to Y$ then clearly $\Sigma \cup \{Y \to R\}$ implies $X \to Y$ as well. Now let $\Sigma \nvDash X \to Y$, so that there exists a relation r on R for which all dependencies in Σ hold, but not $X \to Y$. Then there exist at least two tuples $t_1, t_2 \in r$ with

$$t_1[X] = t_2[X], t_1[Y] \neq t_2[Y]$$

Among all such pairs of tuples, let (t_1, t_2) be one for which the set of attributes $D \subseteq R$ on which t_1 and t_2 differ is minimal. We claim that $r' := \{t_1, t_2\}$ is a relation for which the dependencies in $\Sigma \cup \{Y \to R\}$ hold but not $X \to Y$, thus showing $\Sigma \cup \{Y \to R\} \nvDash X \to Y$.

Clearly $Y \to R$ holds for r', but not $X \to Y$. Furthermore all FDs in Σ hold for r' since $r' \subseteq r$. Now for any join dependency $\bowtie [R_1, \ldots, R_n] \in \Sigma$ let

$$r'' := r'[R_1] \bowtie \ldots \bowtie r'[R_n]$$

be the result of the corresponding project-join mapping of r'. By definition $\bowtie [R_1, \ldots, R_n]$ holds for r' iff $r'' \subseteq r'$.

Let $t_3 \in r''$ be arbitrary. Since $t_1[Y] \neq t_2[Y]$ at least one of the inequalities $t_3[Y] \neq t_1[Y]$ and $t_3[Y] \neq t_2[Y]$ holds, say $t_3[Y] \neq t_1[Y]$. For every attribute $A \in R$ we have $t_3[A] \in \{t_1[A], t_2[A]\}$ by construction of r''. Since $t_1[R \setminus D] = t_2[R \setminus D]$, t_3 differs from t_1 at most on the attributes in D. But t_1, t_2 were chosen to make D minimal, and due to

$$r'' \subseteq r[R_1] \bowtie \ldots \bowtie r[R_n] = r$$

we have $t_3 \in r$. Thus t_3 differs from t_1 (and therefore equals t_2) on all attributes in D. Consequently $t_3 = t_2 \in r'$, which shows $r'' \subseteq r'$ and completes the proof.

Theorem 47 (21). *Let R be a schema with functional and join dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1$ c-dominates $\mathcal{D}_2$, then $\mathcal{D}_1$ weakly csc-dominates $\mathcal{D}_2$.*

Proof. Recall that by definitions 16 and **??** $\mathcal{D}_1$ weakly csc-dominates $\mathcal{D}_2$ iff for every attribute A and every schema R_1 with $A \in R_1 \in \mathcal{D}_1$ there exists a schema $R_2 \in \mathcal{D}_2$ with $A \in R_2$ and $R_1^* \subseteq R_2^*$. As $\mathcal{D}_1$ does not weakly csc-dominate $\mathcal{D}_2$, there must exist A, R_1 with $A \in R_1 \in \mathcal{D}_1$, such that for every schema $R_2 \in \mathcal{D}_2$ with $A \in R_2$ we have $R_1^* \not\subseteq R_2^*$, i.e., $\Sigma \not\models R_2 \to R_1$. To show that $\mathcal{D}_1$ does not c-dominate $\mathcal{D}_2$, we construct a counterexample for any value k.

We construct the counterexample r on R by using the generalized subset construction for $CSC_A(\mathcal{D}_2)$, but for an extended set of constraints

$$\Sigma' := \Sigma \cup \{R_1 \to R\}$$

This makes R_1 a key of R w.r.t. Σ', and by Lemma 46 we still have $\Sigma' \not\models R_2 \to R_1$ for all R_2 in question. Then by Lemma 14 the number of tuples in $r[R_1]$ equals the number of tuples in r, which by Theorem 45 is at least k times larger than the number of tuples in $r[R_2]$, for any $R_2 \in \mathcal{D}_2$ with $A \in R_2$. Thus $\mathcal{D}_1$ does not c-dominate $\mathcal{D}_2$.

The following theorem is well-known [Harary, 1995].

Theorem 48 (Hall's Theorem). *Let M_1, M_2 be finite sets and $\pi : M_1 \to \mathcal{P}(M_2)$ associate a set of permitted values with each element in M_1. Then there exists an injective mapping $f : M_1 \to M_2$ with $f(e) \in \pi(e)$ for all $e \in M_1$ iff for all $m_1 \subseteq M_1$ we have $|m_1| \leq |\pi(m_1)|$, where*

$$\pi(m_1) := \bigcup \{\pi(e) \mid e \in m_1\}$$

COMMENT. We will use Hall's Theorem for multisets rather then sets. Recall that for mappings between multisets we allow different instances of the same value in the source domain to map to different values. This means that for an element e of the source domain, $f(e)$ is a sub-multiset of the target domain, with cardinality equal to the frequency of e in the source domain. Furthermore, we call such a mapping injective if a value in the target domain is mapped to at most as often as it occurs in the target domain.

We can rephrase Hall's theorem for multisets as follows:

Corollary 49. *Let M_1, M_2 be finite multisets and $\pi : M_1 \to \mathcal{P}(M_2)$ associate a set of permitted values with each element in M_1. Then there exists an injective mapping $f : M_1 \to M_2$ with $f(e) \subseteq \pi(e)$ for all $e \in M_1$ iff for all $m_1 \subseteq M_1$ we have $|m_1| \leq |\pi(m_1)|$, where $\pi(m_1)$ is the multiset*

$$\pi(m_1) := \bigcup \{\pi(e) \mid e \in m_1\} \cap M_2$$

Lemma 50. *Let Σ be a set of functional dependencies on R, and $X, Y_1, \ldots, Y_n \subseteq R$. Then $\Sigma \cup \{Y_1 \to R, \ldots, Y_n \to R\} \vDash X \to R$ holds iff $\Sigma \vDash X \to Y_i$ for some $i \in \{1, \ldots, n\}$.*

Proof. (1) If $\Sigma \vDash X \to Y_i$ then $\Sigma \cup \{Y_i \to R\} \vDash X \to R$ by transitivity. (2) Otherwise let X^* denote the closure of X under Σ, and let $r = \{t_1, t_2\}$ be a relation on R containing two tuples with $t_1[X^*] = t_2[X^*]$ and $t_1[A] \neq t_2[A]$ for all $A \notin X^*$. Then Σ holds on r, and since for all Y_i we have $Y_i \not\subseteq X^*$, the functional dependencies $Y_i \to R$ hold as well. It is clear though that $X \to R$ does not hold on r, and thus is not implied by $\Sigma \cup \{Y_1 \to R, \ldots, Y_n \to R\}$.

Theorem 51 (22). *Let R be a schema with functional dependencies Σ, and $\mathcal{D}_1, \mathcal{D}_2$ be decompositions of R. If $\mathcal{D}_1$ dominates $\mathcal{D}_2$, then $\mathcal{D}_1$ strongly csc-dominates $\mathcal{D}_2$.*

Proof. Let $\mathcal{D}_1$ not strongly csc-dominate $\mathcal{D}_2$. This means that for some $A \in R$ there exists no injective mapping

$$f : M_1 := CSC_A(\mathcal{D}_1) \to M_2 := CSC_A(\mathcal{D}_2)$$

with $e \subseteq f(e)$ for all $e \in M_1$. The permitted values for $e \in M_1$ in such a mapping would be

$$\pi(e) := \{e' \in M_2 \mid e \subseteq e'\}$$

Note that M_1, M_2 are multisets, rather than sets. However, to make the formulation of the following arguments easier, we shall regard them as sets by treating multiple occurrences of elements in M_1, M_2 as different. This does not change whether an injective mapping f exists.

By Theorem 48 there exists a set $m_1 \subseteq CSC_A(\mathcal{D}_1)$ with $|m_1| > |m_2|$, where $m_2 := \pi(m_1)$. We now construct a counterexample r on R by using the subset construction for $M_2 \setminus m_2$, but again using an extended set of constraints

$$\Sigma' := \Sigma \cup \{Y \to R \mid Y \in m_1\}$$

This makes all elements in m_1 keys of R w.r.t. Σ', and by Lemma 50 we still have $\Sigma' \not\models X \to R$ for all $X \in M_2 \setminus m_2$. Then by Lemma 14 we have $|r[m_1]| = |m_1| \cdot |r|$ and $|r[m_2]| = |m_2| \cdot |r|$. By Theorem 37 the number $|r|$ of tuples in r is at least k times larger than the number of tuples in $r[X]$, for any $X \in M_2 \setminus m_2$. By choosing k large enough we get

$$|r[M_2 \setminus m_2]| < |r|$$

This gives us

$$\begin{aligned} |r[M_2]| &= |r[m_2]| + |r[M_2 \setminus m_2]| \\ &< |m_2| \cdot |r| + |r| \\ &\leq |m_1| \cdot |r| \\ &= |r[m_1]| \\ &\leq |r[M_1]| \end{aligned}$$

which shows that $\mathcal{D}_1$ does not dominate $\mathcal{D}_2$.

In the last theorem we restricted ourselves to functional dependencies. This is because it does not hold in the presence of multi-valued or join dependencies, as the following example shows.

Example 16. Let $R, \Sigma, \mathcal{D}_1, \mathcal{D}_2$ be as follows:

$$\begin{aligned} &R = ABC, \Sigma = \{A \twoheadrightarrow B\}, \\ &\mathcal{D}_1 = \{AB, AC\}, \mathcal{D}_2 = \{ABC, A\} \end{aligned}$$

It is easy to see that $\mathcal{D}_1$ does not strongly csc-dominate $\mathcal{D}_2$ w.r.t. A:

$$\begin{aligned} CSC_A(\mathcal{D}_1) &= \{AB, AC\} \\ CSC_A(\mathcal{D}_2) &= \{ABC, A\} \end{aligned}$$

It is clear that $\mathcal{D}_1$ dominates $\mathcal{D}_2$ w.r.t. B and C. To show domination w.r.t. attribute count, it thus suffices to prove that for every relation r on R we have

$$count_A(r[\mathcal{D}_1]) \leq count_A(r[\mathcal{D}_2])$$

To do so, we partition r into disjoint relations r_i with $|r_i[A]| = 1$ and $r_i[A] \neq r_j[A]$ for $i \neq j$. Then for each subschema R' of R containing A (i.e., all schemas in $\mathcal{D}_1, \mathcal{D}_2$), $r[R']$ is the disjoint union of the $r_i[R']$, and thus

$$count_A(r[\mathcal{D}_{1/2}]) = \sum count_A(r_i[\mathcal{D}_{1/2}])$$

Therefore we only need to show

$$count_A(r_i[\mathcal{D}_1]) \leq count_A(r_i[\mathcal{D}_2])$$

for all relations r_i. Note that $A \twoheadrightarrow B$ still holds for all r_i, and thus

$$r_i[BC] = r_i[B] \bowtie r_i[C]$$

Abbreviating the cardinalities of $r_i[B], r_i[C]$ with C_B, C_C we obtain

$$|r_i[ABC]| = C_B \cdot C_C, \quad |r_i[A]| = 1$$

This gives us

$$\begin{aligned} count_A(r_i[\mathcal{D}_2]) - count_A(r_i[\mathcal{D}_1]) &= C_B \cdot C_C - (C_B + C_C) + 1 \\ &= (C_B - 1) \cdot (C_C - 1) \\ &\geq 0 \end{aligned}$$

which shows that $\mathcal{D}_1$ dominates $\mathcal{D}_2$ w.r.t. attribute count, even though $\mathcal{D}_1$ does not strongly csc-dominate $\mathcal{D}_2$.

It is an open question how domination w.r.t. size, update time and attribute count can be characterized syntactically in the presence of functional and join dependencies.

4 Relationship to other Normal Forms

A number of normal forms for characterizing well designed relational databases have been proposed, depending on the types of integrity constraints given. For functional dependencies, BCNF and 3NF are the most popular ones. 4NF [Fagin, 1977] is an extension of BCNF for functional and multivalued dependencies. For functional and join dependencies 4NF has been extended to PJ/NF [Fagin, 1979], 5NF [Maier, 1983; Vincent, 1997] and KCNF [Vincent, 1998]. In the following we will compare DNF to existing normal forms for a single schema.

4.1 Lossless DNF and BCNF/4NF/KCNF

Definition 52 (Key-Complete Normal Form). [Vincent, 1998] Let R be a schema with functional and join dependencies Σ. Then R is in *Key-Complete Normal Form (KCNF)*, if for every join dependency $\bowtie [R_1, \ldots, R_n]$ implied by Σ, the keys among $R_1, \ldots, R_n$ contain all attributes in R. That is, we have

$$R = \bigcup \{R_i \in \bowtie [R_1, \ldots, R_n] \mid R_i^* = R\}$$

with $R_i \in \bowtie [R_1, \ldots, R_n]$ meaning $R_i \in \{R_1, \ldots, R_n\}$.

Note that in the case where Σ contains only functional and multivalued dependencies, KCNF is equivalent to 4NF, and when Σ contains only functional dependencies KCNF is equivalent to BCNF [Vincent, 1998].

Given a single schema with constraints Σ, the absence of redundancy, as defined in [Arenas and Libkin, 2003; Vincent, 1998], is precisely characterized by BCNF, 4NF and KCNF. That is, a schema R is free of redundancy iff it is in BCNF, 4NF or KCNF [Arenas and Libkin, 2003; Vincent, 1998].

While our normal form has been designed to minimize size rather than redundancy, the intuition is that minimizing one minimizes the other as well. We will show that this intuition holds in so far, as that a single schema is in KCNF (and thus free of redundancy) iff it is in DNF among all lossless decompositions. Thus lossless DNF can be seen as an extension of BCNF, 4NF and KCNF.

Recall that a decomposition is in DNF if it is minimal *among a given set of "suitable" decompositions*, and that for each such set we may obtain a different DNF. Thus, when we talk about "DNF w.r.t. all lossless decompositions", or "lossless DNF", we mean the version of DNF we obtain when considering a decomposition to be suitable iff it is lossless. By "dependency preserving and lossless DNF", or "dependency preserving DNF" for short, we mean the version of DNF obtained by considering those decompositions as suitable which are both lossless and dependency preserving.

Theorem 53. *Let R be a schema with functional and join dependencies Σ. Then R is in KCNF iff $\{R\}$ is in DNF w.r.t. all lossless decompositions of R.*

Proof. (1) Let R be in KCNF. Let $\mathcal{D} = \{R_1, \ldots, R_n\}$ be any lossless decomposition of R. Then Σ implies the join dependency $\bowtie [R_1, \ldots, R_n]$, and since R is in KCNF, every attribute $A \in R$ lies in some R_i which forms a key of R. Thus $R \in CSC_A(\mathcal{D})$ for all A, so $\{R\}$ strongly csc-dominates $\mathcal{D}$. Therefore $\{R\}$ dominates $\mathcal{D}$ by Theorem 20. As this holds for all lossless decompositions $\mathcal{D}$, $\{R\}$ is in DNF.

(2) Let R not be in KCNF. Then Σ implies a join dependency $\bowtie [R_1, \ldots, R_n]$ such that

$$\bigcup \{R_i \mid \Sigma \vDash R_i \rightarrow R\} \neq R$$

The decomposition $\mathcal{D} = \{R_1, \ldots, R_n\}$ is lossless, and there exists an attribute $A \in R$ which does not lie in any R_i which forms a key of R. Clearly $\mathcal{D}$ weakly csc-dominates $\{R\}$, and since $R \notin CSC_A(\mathcal{D})$ we have that $\{R\}$ does not weakly csc-dominate $\mathcal{D}$. Thus $\mathcal{D}$ strictly c-dominates $\{R\}$ by theorems 20 and 21, showing that $\{R\}$ is not in DNF.

Note that, while lossless DNF and KCNF are the same for a single schema, they differ significantly when applied to multiple schemas.

4.2 Dependency Preserving DNF and 3NF/EKNF

When considering only dependency preserving decompositions, we can only hope to compare DNF with normal forms which always allow dependency preserving decompositions (i.e., not BCNF). The most well-known normal form with this property is 3NF. However, as was pointed out in [Zaniolo, 1982], 3NF does not always enforce beneficial decomposition, even though they may not cause any loss of dependencies. The following example, taken from [Zaniolo, 1982], illustrates this.

Example 17. Let $R = ABC$ and $\Sigma = \{A \to B, B \to A\}$. Then AC and BC are minimal keys of R, and thus all attributes are prime. Therefore R is already in 3NF, even though dependency preserving decompositions exist, such as $\{AB, BC\}$ or $\{AB, AC\}$.

As an improvement, the authors of [Zaniolo, 1982] suggest a new normal form which is stronger than 3NF but still allows dependency preserving decompositions. They strengthen 3NF by allowing as RHS of a non-key FD only those prime attributes, which appear in the LHS of an atomic key dependency. Note that atomic FDs are called *elementary* in [Zaniolo, 1982].

Definition 54 (Elementary). [Zaniolo, 1982] Let R be a schema with FDs Σ. A FD $X \to A$ is called *elementary* if Σ^* contains no FD $X' \to A$ with $X' \subsetneq X$. A key is elementary if it forms the LHS of an elementary FD. An attribute is an elementary key attribute if it lies in an elementary key of R.

Definition 55 (Elemental Key Normal Form). [Zaniolo, 1982] Let R be a schema with FDs Σ. Then R is in *elemental key normal form* (EKNF) if for every non-trivial FD $X \to A$ on R

(a) X is a key of R, or

(b) A is an elementary key attribute for R.

Note that the schema R from example 17 is not in EKNF, since neither A nor B are elementary key attributes. Thus EKNF may enforce useful decomposition which 3NF does not.

Lemma 56. *Let R be a single schema with FDs Σ. If R is in dependency preserving DNF, then it is also in EKNF.*

Proof. Assume that R is not in EKNF. Then there exists a FD $X \to A \in \Sigma^{*a}$ such that X is not a key of R, and A does not lie in the LHS of any key FD in Σ^{*a}. Furthermore, R is strictly c-dominated by the decomposition

$$\mathcal{D} := \{R \setminus A\} \cup \{S \subsetneq R \mid S \text{ is not a key of } R\}$$

since A does not lie in $R \setminus A$, which is the only key schema in $\mathcal{D}$.

It remains to show that $\mathcal{D}$ is dependency preserving. Clearly the only FDs in Σ^{*a} which do not lie in $\Sigma^*[\mathcal{D}]$ are key FDs containing A. They must be of the form $Y \to A$, since A does not lie in the LHS of any key FD. However, the FDs $Y \to X$ and $X \to A$ both lie in $\Sigma^*[\mathcal{D}]$, and together they imply $Y \to A$.

As EKNF implies 3NF [Zaniolo, 1982], this provides us with a comparison of dependency preserving DNF and 3NF as well.

5 Computing Domination Normal Form

Having defined when schemas are in DNF, we are now looking for an algorithm which, given a schema R with constraints Σ, computes a decomposition of R which is in DNF. For this we will restrict ourselves to the case where the only constraints given are functional dependencies.

Lemma 57. *Let R be a schema with constraints Σ, and $\mathcal{D}$ be a decomposition of R. If $\mathcal{D}$ contains two different $R_1, R_2 \in \mathcal{D}$ with $R_1^* = R_2^*$, then*

$$\mathcal{D}' := \mathcal{D} \setminus \{R_1, R_2\} \cup \{R_1 \cup R_2\}$$

dominates $\mathcal{D}$. If $R_1 \cap R_2 \neq \emptyset$, then $\mathcal{D}'$ strictly dominates $\mathcal{D}$.

Proof. For every relation r on R the projections $r[R_1], r[R_2]$ can be obtained by projecting from $r[R_1 \cup R_2]$. As R_1 and R_2 are keys of

$$R_1 \cup R_2 \subseteq R_1^* = R_2^*$$

no tuples are lost in this projection. Thus the size of $\mathcal{D}$ and $\mathcal{D}'$ varies by the size for the values of attributes in $R_1 \cap R_2$, as those are stored twice in $\mathcal{D}$.

We shall define equivalence classes for functional dependencies, as well as for schemas.

Definition 58 (Equivalence/Higher Order). Let R be a schema with constraints Σ and $X, Y \subseteq R$. We call X and Y *equivalent* if $X^* = Y^*$. We say that X is of *higher order* than Y if $X^* \supseteq Y^*$.

We call two functional dependencies $X_1 \to Y_1, X_2 \to Y_2$ implied by Σ equivalent if their left hand sides X_1 and X_2 are equivalent (and similarly for higher order). When partitioning a set Σ of FDs into equivalence classes, we denote them by

$$EQ_X := \{Y \to Z \in \Sigma \mid Y^* = X^*\}$$

and call X^* the closure of EQ_X.

This groups schemas and functional dependencies into equivalence classes. As there is an obvious correspondence between equivalence classes of schemas and those of functional dependencies, we will not always distinguish between them, and will compare equivalence classes w.r.t. higher order as well.

When searching for a DNF decomposition, Lemma 57 tells us that it suffices to only consider decompositions which contain at most one schema for each equivalence class of schemas.

Definition 59 (Higher Order Schema/Attribute). Let $\mathcal{D}$ be a decomposition of R and $X \subseteq R$. We define the *higher order schemas* and *higher order attributes* of X in $\mathcal{D}$ as

$$HOS_X(\mathcal{D}) := \{R_j \in \mathcal{D} \mid X^* \subseteq R_j^*\}$$
$$HOA_X(\mathcal{D}) := \bigcup HOS_X(\mathcal{D})$$

Similarly, the *strictly higher order schemas* and *strictly higher order attributes* of X in $\mathcal{D}$ are

$$SHOS_X(\mathcal{D}) := \{R_j \in \mathcal{D} \mid X^* \subsetneq R_j^*\}$$
$$SHOA_X(\mathcal{D}) := \bigcup SHOS_X(\mathcal{D})$$

Lemma 60. *Let $\mathcal{D}_1, \mathcal{D}_2$ be two decompositions of R. Then $\mathcal{D}_1$ weakly csc-dominates $\mathcal{D}_2$ iff for every schema $X \in \mathcal{D}_1$ we have*

$$X \subseteq HOA_X(\mathcal{D}_2)$$

Proof. (1) By definition $\mathcal{D}_1$ weakly csc-dominates $\mathcal{D}_2$ iff for every attribute $A \in R$ and every $R_1^* \in CSC_A(\mathcal{D}_1)$ there exists $R_2^* \in CSC_A(\mathcal{D}_2)$ with $R_1^* \subseteq R_2^*$. In other words, for every pair (A, R_1) with $A \in R_1 \in \mathcal{D}_1$ there exists R_2 with $A \in R_2 \in \mathcal{D}_2$ and $R_1^* \subseteq R_2^*$.

(2) Furthermore we have $X \subseteq HOA_X(\mathcal{D}_2)$ for all $X \in \mathcal{D}_1$ iff for every pair (A, X) with $A \in X \in \mathcal{D}_1$ there exists $Y \in HOS_X(\mathcal{D}_2)$ with $A \in Y$, that is Y with $A \in Y \in \mathcal{D}_2$ and $X^* \subseteq Y^*$.

Using (1) and (2) together (with $X = R_1$ and $Y = R_2$) we obtain the claim.

Definition 61. Let R be a schema with FDs Σ, and $\mathcal{D}$ a set of subschemas of R. We denote the set of all FDs in Σ which lie in schemas in $\mathcal{D}$ by

$$\Sigma[\mathcal{D}] := \bigcup_{R_j \in \mathcal{D}} \Sigma[R_j]$$

5.1 Dependency Preserving DNF

We are now able to construct an algorithm to compute a DNF decomposition for the case where Σ contains only functional dependencies, and where we consider

only lossless and dependency preserving decompositions. For this, we need some definition and properties of partial covers from [Köhler, 2008], rephrased slightly to suit our needs.

Definition 62 (Partial Cover). Let Σ be a set of FDs, and $E_X \subseteq \Sigma$ be an equivalence class of Σ. A set $C \subseteq \Sigma^*$ is a *partial cover* of E_X (w.r.t. Σ) if

$$C[X^*] \cup (\Sigma \setminus E_X)$$

is a cover of Σ.

Lemma 63. *[Köhler, 2008] Let Σ be a set of FDs, and let EQ be the partition of Σ into equivalence classes. Then a set C of FDs is a cover of Σ iff C is a partial cover (w.r.t. Σ) for all equivalence classes $EQ_j \in EQ$.*

Lemma 64. *Let R be a schema with FDs Σ, and $\mathcal{D}$ a dependency preserving decomposition of R. Let further EQ_X be an equivalence class of Σ. Then*

$$\Sigma^{*a}[HOS_X(\mathcal{D})] \subseteq \Sigma^{*a}[HOA_X(\mathcal{D})]$$

and each of the two dependency sets forms a partial cover of EQ_X.

Proof. Since $\mathcal{D}$ is dependency preserving, $\Sigma^{*a}[\mathcal{D}]$ must form a cover of Σ, and thus a partial cover of EQ_X. By Definition 62 the only FDs of interest for forming a partial cover of EQ_X are those LHS-equivalent to the FDs in EQ_X. All of them lie in schemas R_j with $X^* \subseteq R_j^*$, so

$$\Sigma^{*a}[\{R_j \in \mathcal{D} \mid X^* \subseteq R_j^*\}]$$

already forms a partial cover of EQ_X.

We use this to synthesize a dependency preserving decomposition as follows.

Algorithm "dependency preserving DNF decomposition"

INPUT: schema R, canonical cover Σ
OUTPUT: decomposition $\mathcal{D}$ in DNF

$\mathcal{D} := \emptyset$
if Σ contains no key dependencies then
  add minimal key R_{key} of R to $\mathcal{D}$
partition Σ into equivalence classes EQ
while $EQ \neq \emptyset$ do
    pick maximal $EQ_j \in EQ$ and remove it from EQ
    $R_j :=$ closure of FDs in EQ_j
    $A_{\mathcal{D}} := SHOA_{R_j}(\mathcal{D})$

first for all $A \in R_j \setminus A_{\mathcal{D}}$ and then for all $A \in R_j \cap A_{\mathcal{D}}$ do
if $\Sigma^{*a}[\mathcal{D} \cup \{R_j \setminus A\}]$ is a partial cover of EQ_j then
$R_j := R_j \setminus A$
end
if $R_j \neq \emptyset$ then
$\mathcal{D} := \mathcal{D} \cup \{R_j\}$
end

Theorem 65. *Algorithm "dependency preserving DNF decomposition" returns a lossless, dependency preserving DNF decomposition of R.*

Proof. (1) We first show that $\mathcal{D}$ is dependency preserving and lossless. This is the case iff $\Sigma^{*a}[\mathcal{D}]$ is a cover of Σ, and $\mathcal{D}$ contains a key of R. If Σ contains no key dependency, then a minimal key R_{key} of R is added to $\mathcal{D}$. Otherwise it suffices to show that $\Sigma^*[\mathcal{D}]$ is a cover of Σ, since this implies that $\mathcal{D}$ contains a key of R. In each iteration of the while loop, it is ensured that for the decomposition $\mathcal{D}$ computed so far, $\Sigma^*[\mathcal{D}]$ forms a partial cover for the equivalence classes removed from EQ. After processing all equivalence classes, we therefore obtain a decomposition for which $\Sigma^*[\mathcal{D}]$ covers Σ.

It remains to show that $\mathcal{D}$ is not strictly dominated or c-dominated by any other lossless and dependency preserving decomposition $\mathcal{D}'$ of R.

(2) We start by proving that $\mathcal{D}'$ does not strictly dominate $\mathcal{D}$. Consider the schemas $R_1, \ldots, R_n \in \mathcal{D}$ (including R_{key} if it was added) in the order as they were added to $\mathcal{D}$ (with indices describing this order). Let R_k be the first such schema which is not contained in $\mathcal{D}'$ (if all R_j are contained in $\mathcal{D}'$ then clearly $\mathcal{D}$ dominates $\mathcal{D}'$), and $C := R_k^*$ its closure. By Lemma 64 the set $\Sigma^*[HOS_C(\mathcal{D}')]$ forms a partial cover of EQ_k. Let further

$$H'_k := \bigcup (HOS_C(\mathcal{D}') \setminus \{R_1, \ldots, R_{k-1}\})$$

so that $\Sigma^*[\{R_1, \ldots, R_{k-1}\} \cup H'_k]$ forms a partial cover of EQ_k. Since R_k was constructed as minimal such that $\Sigma^*[R_1, \ldots, R_k]$ forms a partial cover of EQ_k, $H'_k \subsetneq R_k$ cannot hold. If $H_k = R_k$ then all schemas in $HOS_C(\mathcal{D}') \setminus \mathcal{D}$ must be of order EQ_k. By Lemma 57 we may assume that this does not happen, so $H_k \not\subseteq R_k$. Thus there exists at least one attribute A which lies in some schema

$$R'_A \in HOS_C(\mathcal{D}') \setminus \{R_1, \ldots, R_{k-1}\}$$

but not in R_k.

Consider containing schema closures $CSC_A(\mathcal{D})$ and $CSC_A(\mathcal{D}')$. If $\mathcal{D}'$ were to dominate $\mathcal{D}$, then $CSC_A(\mathcal{D}')$ would strongly inclusion dominate $CSC_A(\mathcal{D})$.

Equivalently, $CSC_A(\mathcal{D}' \setminus \{R_1, \ldots, R_{k-1}\})$ would have to strongly inclusion dominate $CSC_A(\mathcal{D} \setminus \{R_1, \ldots, R_{k-1}\})$. However, we have

$$R'^*_A \in CSC_A(\mathcal{D}' \setminus \{R_1, \ldots, R_{k-1}\})$$

but no schema in $CSC_A(\mathcal{D} \setminus \{R_1, \ldots, R_{k-1}\})$ includes R'^*_A. This is a contradiction, which shows that $\mathcal{D}'$ does not dominate $\mathcal{D}$.

(3) Finally, we need to show that $\mathcal{D}'$ does not strictly c-dominate $\mathcal{D}$. Assume the contrary, so that by Lemma 60 we have for all $R' \in \mathcal{D}'$ that

$$R' \subseteq HOA_{R'}(\mathcal{D}),$$

and there exist a schema $R_w \in \mathcal{D}$ for which

$$R_w \nsubseteq HOA_{R_w}(\mathcal{D}')$$

Let R_w be the first such schema in the sequence of schemas $R_1, \ldots, R_n \in \mathcal{D}$, and let $C := R^*_w$ be its closure. Then for every $R_j \in SHOS_C(\mathcal{D})$ we get

$$R_j \subseteq HOA_{R_j}(\mathcal{D}') \subseteq SHOA_C(\mathcal{D}')$$

and thus

$$SHOA_C(\mathcal{D}) \subseteq SHOA_C(\mathcal{D}')$$

Inclusion in the opposite direction holds by similar argument, showing

$$SHOA_C(\mathcal{D}) = SHOA_C(\mathcal{D}')$$

This attribute set is computed as the set $A_\mathcal{D}$ during the construction of R_w:

$$A_\mathcal{D} = SHOA_C(\{R_1, \ldots, R_{w-1}\}) = SHOA_C(\mathcal{D}) = SHOA_C(\mathcal{D}')$$

Since $R_w \nsubseteq HOA_{R_w}(\mathcal{D}') \supseteq A_\mathcal{D}$ we have $R_w \nsubseteq A_\mathcal{D}$. As we tried removing attributes outside $A_\mathcal{D}$ first when constructing R_w, the set

$$\Sigma^*[A_\mathcal{D}] = \Sigma^*[SHOA_C(\mathcal{D}')]$$

cannot form a partial cover of EQ_w. By Lemma 64 the set $\Sigma^*[HOA_C(\mathcal{D}')]$ does form a partial cover of EQ_k, so $\mathcal{D}'$ must contain at least one schema R'_w with $R'^*_w = C$. By Lemma 57 we may assume that R'_w is the only such schema in $\mathcal{D}'$. We have by assumption that

$$R'_w \subseteq HOA_C(\mathcal{D}) = SHOA_C(\mathcal{D}) \cup R_w = A_\mathcal{D} \cup R_w$$

and thus

$$R'_w \setminus A_\mathcal{D} \subseteq R_w \setminus A_\mathcal{D}$$

Furthermore, we can split $HOS_C(\mathcal{D}')$ into $SHOS_C(\mathcal{D}')$ and R'_w, and get (using Lemma 64 once more) that

$$\Sigma^*[A_\mathcal{D} \cup R'_w] \supseteq \Sigma^*[SHOS_C(\mathcal{D}') \cup \{R'_w\}] = \Sigma^*[HOS_C(\mathcal{D}')]$$

must be a partial cover of EQ_w. However, by trying to remove attributes not in $A_\mathcal{D}$ first, we constructed R_w such that $R_w \setminus A_\mathcal{D}$ is minimal with $\Sigma^*[A_\mathcal{D} \cup R_w]$ being a partial cover for EQ_w (note that $\Sigma^*[\{R_1, \ldots, R_{w-1}\}] \subseteq \Sigma^*[A_\mathcal{D}]$) . Thus $R'_w \setminus A_\mathcal{D}$ cannot be a proper subset of $R_w \setminus A_\mathcal{D}$, which gives us

$$R'_w \setminus A_\mathcal{D} = R_w \setminus A_\mathcal{D}$$

But this means that

$$R_w \subseteq R'_w \cup A_\mathcal{D} = R'_w \cup SHOA_C(\mathcal{D}') = HOA_C(\mathcal{D}')$$

which contradicts our initial assumption for R_w.

5.2 Local DNF

One potential complaint about the global optimization approach taken by DNF, is that it does not enforce local optimization. That is to say, even if a decomposition is in DNF, each of its schemas taken individually does not have to be. Despite this, our algorithm always produces DNF decompositions which are also "locally optimal".

Theorem 66. *Let $\mathcal{D}$ be a decomposition produced by algorithm "dependency preserving DNF decomposition". Then every schema $R_X \in \mathcal{D}$ is in dependency preserving DNF w.r.t. $\Sigma^*[R_X]$.*

Proof. Let $\mathcal{D}_X$ be any dependency preserving decomposition of R_X. Then $\mathcal{D}_X$ is dominated by the single schema

$$R'_X := \bigcup \{R_j \in \mathcal{D}_X \mid R_j \text{ is a key of } R_X\}$$

Clearly R'_X preserves all key FDs in $\Sigma^*[\mathcal{D}]$. Thus $EQ_X[\mathcal{D}_X] \subseteq EQ_X[R'_X]$, so $EQ_X[R'_X]$ implies $EQ_X[R_X]$. However, R_X has been constructed minimal such that $EQ_X[R_X]$ has some partial cover property. Thus $R'_X = R_X$, which shows that R_X dominates every dependency preserving decomposition $\mathcal{D}_X$. It follows that R_X is in dependency preserving DNF.

Corollary 67. *Algorithm "dependency preserving DNF decomposition" produces a decomposition into EKNF.*

We note that this result is only due to our construction method, i.e., dependency preserving DNF does not imply EKNF in general.

Example 18. Let $R = ABCD$ and $\Sigma = \{A \to B, B \to C, CD \to A\}$. Then the decomposition $\mathcal{D} = \{ABC, ACD\}$ is in dependency preserving DNF (note that it is *not* strictly c-dominated by $\{AB, BC, ACD\}$ since C already appears in the key schema ACD). However, $\mathcal{D}$ is not in EKNF, since ABC contains $B \to C$.

6 Conclusion

We have introduced a new normal form called DNF for relational databases, based on the requirements for a decomposition. Here, a decomposition is in DNF if and only if there is no "better" decomposition among the decompositions considered. The partial orders describing this "better" property have been defined in semantic terms, and for functional and in part for join dependencies, they have been characterized syntactically. Using this syntactical characterization we then showed that, when considering all lossless decompositions, our normal form is an extension to the existing normal forms BCNF, 4NF and KCNF, which have already proven to be useful over the past 30 years.

For multiple schemas DNF appears more suitable than just considering schemas individually, as has been done traditionally. At the same time it is always applicable, in that a decomposition into DNF always exists, even when we restrict ourselves to certain types of decompositions, e.g. dependency preserving ones.

Finally, we provided an algorithm which always finds lossless and dependency preserving decompositions into DNF, and showed that these decompositions are even locally optimal. We conclude by mentioning a number of open problems.

- Theorem 22 is restricted to functional dependencies, and does not hold in the presence of multivalued or join dependencies. It would be interesting to know what the correct syntactic characterization of DNF would be here.
- Our definition of DNF focuses on minimizing size or update time, with the intuition that this could minimize redundancy as well. Alternatively one could attempt to minimize redundancy directly, and it would be interesting to see whether the results are the same.
- One could try to achieve other desirable properties for a decomposition, such as e.g. acyclicity [Beeri et al., 1983]. This has no impact on the orders for comparing decompositions, but could lead to different decomposition algorithms.
- While we have an algorithm for computing a decomposition into dependency preserving DNF, we currently lack such an algorithm for lossless DNF.
- Our assumption that all domains are infinite and unbounded is not always realistic. It is not clear yet how DNF w.r.t. size could be defined suitably for

finite or bounded domains. The current definition fails, since (almost) every decomposition c-dominates every other.

– Finally, it is unclear how to test whether a given decomposition is in DNF, and how difficult such a test is. We suspect that it is at least co-NP hard.

References

[Arenas and Libkin, 2003] Arenas, M. and Libkin, L. (2003). An information-theoretic approach to normal forms for relational and XML data. In *PODS*, pages 15–26.

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

[Beeri et al., 1983] Beeri, C., Fagin, R., Maier, D., and Yannakakis, M. (1983). On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513.

[Biskup, 1995] Biskup, J. (1995). Achievements of relational database schema design theory revisited. In Libkin, L. and Thalheim, B., editors, *Semantics in Databases*, volume 1358 of *Lecture Notes in Computer Science*, pages 29–54. Springer.

[Casanova et al., 1982] Casanova, M. A., Fagin, R., and Papadimitriou, C. H. (1982). Inclusion dependencies and their interaction with functional dependencies. In *PODS '82*, pages 171–176, New York, NY, USA. ACM Press.

[Fagin, 1977] Fagin, R. (1977). Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst.*, 2(3):262–278.

[Fagin, 1979] Fagin, R. (1979). Normal forms and relational database operators. In Bernstein, P. A., editor, *SIGMOD Conference*, pages 153–160. ACM.

[Harary, 1995] Harary, F. (1995). *Graph Theory*. Addison-Wesley.

[Köhler, 2007] Köhler, H. (2007). Domination Normal Form: decomposing relational database schemas. In *ACSC '07: Proceedings of the thirtieth Australasian conference on Computer science*, pages 79–85, Ballarat, Australia. Australian Computer Society, Inc.

[Köhler, 2008] Köhler, H. (2008). Autonomous sets - a method for hypergraph decomposition with applications in database theory. In Hartmann, S. and Kern-Isberner, G., editors, *FoIKS*, volume 4932 of *Lecture Notes in Computer Science*, pages 78–95. Springer.

[Levene and Loizou, 1999] Levene, M. and Loizou, G. (1999). *A Guided Tour of Relational Databases and Beyond.* Springer.

[Maier, 1983] Maier, D. (1983). *The Theory of Relational Databases.* Computer Science Press.

[Mannila and Räihä, 1987] Mannila, H. and Räihä, K.-J. (1987). *The Design of Relational Databases.* Addison-Wesley.

[Vincent, 1997] Vincent, M. W. (1997). A corrected 5NF definition for relational database design. *Theor. Comput. Sci.*, 185(2):379–391.

[Vincent, 1998] Vincent, M. W. (1998). Redundancy elimination and a new normal form for relational database design. In *Semantics in Databases*, volume 1358 of *Lecture Notes in Computer Science*, pages 247–264. Springer.

[Wang and Topor, 2005] Wang, J. and Topor, R. W. (2005). Removing XML data redundancies using functional and equality-generating dependencies. In Williams, H. E. and Dobbie, G., editors, *ADC*, volume 39 of *CRPIT*, pages 65–74. Australian Computer Society.

[Zaniolo, 1982] Zaniolo, C. (1982). A new normal form for the design of relational database schemata. *ACM Trans. Database Syst.*, 7(3):489–499.