Decomposition by Pivoting and Path Cardinality Constraints

Sven Hartmann

FB Mathematik, Universität Rostock, 18051 Rostock, Germany

March 16, 2000

Abstract. In the relational data model, the problem of data redundancy has been successfully tackled via decomposition. In advanced data models, decomposition by pivoting provides a similar concept. Pivoting has been introduced by Biskup et al. [5], and used for decomposing relationship types according to a unary nonkey functional dependency. Our objective is to study pivoting in the presence of cardinality constraints which are commonly used in semantic data models.
In order to ensure the equivalence of the given schema and its image under pivoting, the original application-dependent constraints have to be preserved. We discuss this problem for sets of participation and co-occurrence constraints. In particular, we prove the necessity of path cardinality constraints, and give an appropriate foundation for this concept.

1 Introduction

Database modeling aims on designing efficient and appropriate databases. It is widely accepted now that databases are best designed first on a conceptual level. The result of this process is a conceptual schema which describes the requirements that the desired database must achieve. Usually, conceptual design does not consist of a single design step, but of a step-by-step process. Each step refines the schema, e.g. by adding information, changing the structure of the schema, refining types or generating subtypes. Composition, extension and decomposition are well-known operations to refine types during the design process.

In this paper, we are almost exclusively concerned with decomposition. In the relational model, this transformation has been successfully used e.g. for reducing redundancy, splitting overloaded concepts or making constraint enforcement more efficient. There is a fairly extensive literature on this strategy, to which the reader may wish to turn for details, e.g. [2, 18, 23]. The same idea may be used in semantic data models which are frequently used in conceptual design. Decomposition primitives in the entity-relationship model have been discussed e.g. in [2, 3, 22].

An interesting, new approach towards decomposition in semantic data models, was suggested by Biskup et al. [5, 6]. They introduced the operation of pivoting for decomposing relationship types. Pivoting separates apart some components of a relationship type and assembles them in a newly generated type. Afterwards, the new type is linked to the old one via a pivot component.

In the relational case, decomposition is usually motivated by functional dependency. In particular, normalization theory up to BCNF is based on this idea. In semantic data models, however, this concept falls short. During the last few decades, the area of integrity constraints has attracted considerable research interest. A large amount of different constraint classes has been discussed in the literature and actually used in database design.

Cardinality constraints are among the most popular classes of integrity constraints. They impose lower and upper bounds on the number of relationships an entity of a given type may be involved in. For cardinality constraints and generalizations, see [16, 22]. The objective of this paper is to study pivoting in the presence of cardinality constraints. Our investigation is not restricted to ordinary cardinality constraints, but we also consider co-occurrence constraints, which among others include functional and numerical dependencies.

The paper is organized as follows. Section 2 describes the data model to be used. Though cardinality constraints have also been studied in object-oriented context, they are most popular in semantic data models. Therefore, all our considerations are carried out in an extended entity-relationship context. In Section 3, we briefly review decomposition under functional dependencies and highlight the idea of pivoting. Section 4 gives a formal definition of cardinality constraints. In Section 5 we give a generalized definition of pivoting, and show how to use pivoting under cardinality constraints. In Section 6, we discuss the problem of preserving constraints. We point out that the usual concept of cardinality constraints is not sufficient to cope with this problem. Finally, in Section 7, we identify conditions for a schema being equivalent to its pivoted version, namely the expressive power of path cardinality constraints.

2 The data model to be used

The entity-relationship approach to conceptual design provides a simple way of representing data in the form of entities and relationships among them. In this section, we shall briefly review basic concepts of this approach.

Let $\underline{E}$ be a finite set, whose elements are called *entity types*. With each entity type $\underline{e}$ we associate a finite set $\underline{e}^t$ called the population of the type $\underline{e}$ (at moment t). The members of $\underline{e}^t$ are entity instances or entities, for short. Intuitively, entities may be seen as real-world objects which are of interest for the application under consideration. By classifying them and specifying their significant properties (attributes), we obtain entity types which are frequently used to model the objects in their domains.

A *(first-order) relationship type* $\underline{r}$ is a set $\{\underline{e}_1, \ldots, \underline{e}_n\}$ of elements from $\underline{E}$. Relationship types are used to model associations between real-world objects, i.e. entities. A relationship or instance of type $\underline{r}$ is a tuple from the cartesian product $\underline{e}_1^t \times \ldots \times \underline{e}_n^t$. A finite collection $\underline{r}^t$ of such relationships forms the population of $\underline{r}$ at moment t. Analogously, higher-order relationship types may be defined hierarchically, cf. [21]. Given some entity and/or relationship types, a set $\{\underline{u}_1, \ldots, \underline{u}_n\}$ of them forms a *(higher-order) relationship type.* As above, we define relationships of type $\underline{r}$ as tuples from the cartesian product $\underline{u}_1^t \times \ldots \times \underline{u}_n^t$ at moment t.

In a relationship type $\underline{r} = \{\underline{u}_1, \ldots, \underline{u}_n\}$ each of the entity or relationship types $\underline{u}_1, \ldots, \underline{u}_n$ is said to be a *component type* of $\underline{r}$. Additionally, each of the pairs $(\underline{r}, \underline{u}_j)$ is called a *link.*

A finite set $\underline{D} = \{\underline{r}_1, \ldots, \underline{r}_k\}$ of entity and relationship types used to represent information is called a *database schema.* Of course, we assume that with each relationship type $\underline{r}$, all its component types belong to $\underline{D}$, too. When replacing each type $\underline{r}$ in $\underline{D}$ by its population $\underline{r}^t$ at moment t, we obtain a database or instance $\underline{D}^t$ of $\underline{D}$.

One of the strengths of the entity-relationship approach is its ability to provide a simple graphical representation of information. Consider the digraph $\mathfrak{ERD}$ with vertex set $\underline{D}$ whose arcs are just the links of the database schema. This digraph is known as the *entity-relationship diagram* of $\underline{D}$. It is usual to represent entity types graphically by rectangles and relationship types by diamonds.

Example. Below we give a small motivating example to illustrate the issues we are going to tackle. Consider a part of a university database schema involving the entity types PROFESSOR, TUTOR, COURSE, BUILDING, HALL and DATE. On this set we define a relationship type LECTURE reflecting assignments of professors and their tutors to courses in combination with the dates and locations they take place at.

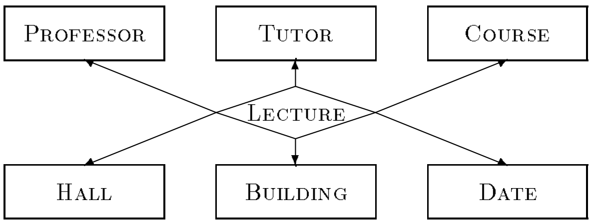

Fig. 1. Entity-relationship diagram for our example.

Some designers would perhaps tend to use smaller relationships. This is an issue where the designer must rely one his personal judgement and experience. However, in conceptual design, a designer should be supported to concentrate on the essential parts of the application: he has to determine the objects of interest and to specify their associations. In early design steps, he should not be confronted with restrictions, or to take into account the features of future design steps, such as breaking up overloaded relationships.

3 Decomposition by pivoting

In database design, great attention is devoted to the modeling of semantics. Central to this idea is the notion of integrity constraints. Defining and enforcing integrity constraints helps to guarantee that the database correctly reflects the underlying domain of interest. Functional dependencies are commonly recognized as the most fundamental constraints. They have been introduced in the relational model by Codd [9], and later extended to other data models, cf. [21, 24]. Among functional dependencies, key dependencies are of utmost interest, as they help to identify the tuples in a population.

Consider an instance r of the relationship type $\underline{r}$. By $r[X]$ we denote the restriction of r to a subset $X \subseteq \underline{r}$. Moreover, for a population $\underline{r}^t$, let $\pi_X(\underline{r}^t) = \{r[X] : r \in \underline{r}^t\}$ denote its projection to X. A subset $X \subseteq \underline{r}$ is a *key* for $\underline{r}$ if the restrictions $r[X]$, $r \in \underline{r}^t$, are pairwise distinct. A database instance $\underline{D}^t$ satisfies the *key dependency* $\underline{r} : X \to \underline{r}$, when X is a key for $\underline{r}$.

A *functional dependency* is an expression $\underline{r} : X \to Y$, where X and Y are subsets of $\underline{r}$. A database instance $\underline{D}^t$ satisfies this constraint, if we have $r_1[Y] = r_2[Y]$ whenever $r_1[X] = r_2[X]$ holds for any two relationships r_1 and r_2 in $\underline{r}^t$. A functional dependency is said to be *unary*, if X contains a single component type from $\underline{r}$.

Example. In the university schema, constraints are defined to meet the requirements of the university schedule. Let us consider some examples. If every professor has only one tutor per course, then the functional dependency LECTURE: {PROFESSOR, COURSE} $\to$ {TUTOR} holds. Since every lecture hall is situated in only one building, we also have the unary functional dependency LECTURE: {HALL} $\to$ {BUILDING}.

Constraints are to be considered as an integral part of a database schema. We use the notion $(\underline{D}, \Sigma)$ to refer to a schema $\underline{D}$ together with a set Σ of constraints defined on $\underline{D}$. A database instance $\underline{D}^t$, satisfying each constraint in Σ, is said to be *legal* for Σ.

Let Σ_1 and Σ_2 be two constraint sets declared on $\underline{D}$. Then Σ_1 *implies* Σ_2 (denoted by $\Sigma_1 \models \Sigma_2$) if every legal database instance $\underline{D}^t$ for Σ_1 is also legal for Σ_2. Both Σ_1 and Σ_2 are *semantically equivalent* if we have $\Sigma_1 \models \Sigma_2$ as well as $\Sigma_2 \models \Sigma_1$.

If not all instances are legal, it stands to reason that there appears some kind of redundancy in legal instances. This problem and its consequences have been widely studied in literature, for a survey see [18, 23]. In the relational model, redundancy has been successfully tackled via decompositions. To be brief, a decomposition of a relation type R is its replacement by two subsets X and Y such that $X \cup Y = R$ holds. In general, we only consider lossless decompositions: every legal population of R must be recoverable, i.e. the natural join of its projections $\pi_X(R^t)$ and $\pi_Y(R^t)$. It is well-known that this property holds if and only if one of the functional dependencies $R : X \cap Y \to X$ or $R : X \cap Y \to Y$ holds for R.

It is natural to ask whether this idea can be extended to advanced data models, such as semantic or object-oriented ones. In [5], Biskup et al. gave an interesting answer to this question. They suggest the transformation of pivoting to decompose relationship types. Assume, we are given a unary functional dependency $\underline{r} : \{\underline{u}\} \to Y$. Then, pivoting separates apart the component types in Y into a newly generated relationship type. The new type is declared to be a subtype of the so-called pivot component $\underline{u}$. Moreover the pivot component provides the link between the original relationship type and the newly generated one. For further details on this approach, we refer to [4–6].

The above discussion gives only an informal account of decomposition in database design. For a rigorous treatment, the reader should consult e.g. [18]. It is noteworthy, that eliminating redundancy is by far not the only motivation for performing decompositions: a schema where the relationship types are too large makes it necessary to use a single tuple for storing several facts. Such a schema is said to suffer from update anomalies. Moreover, decompositions are of considerable interest when translating conceptual schemas to target models such as object-oriented models. For a discussion of this issue, we refer to [5].

4 Cardinality constraints

Cardinality constraints are often regarded as one of the basic constituents of the entity-relationship model. They are already present in Chen's seminal paper [8], and have been frequently used in conceptual design since then. A *cardinality constraint* is an expression $card(\underline{r}, \underline{u}) = (a, b)$, where $\underline{r}$ is a relationship type, and $\underline{u}$ is one of its component types. A database instance $\underline{D}^t$ satisfies this constraint if every instance of $\underline{u}$ appears in at least a and at most b relationships $r \in \underline{r}^t$. Often, cardinality constraints are reflected graphically in the entity-relationship diagram as (a, b)-labels associated to the corresponding links, see Figure 2.

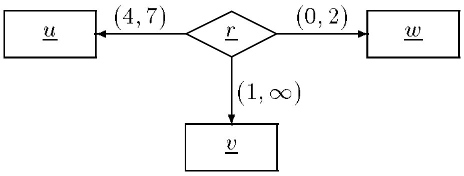

Fig. 2. An entity-relationship diagram with cardinality constraints.

Cardinality constraints are available in most semantic data models, but also in object-oriented models [7, 10]. For a survey on cardinality constraints, the interested reader is referred to [16, 22]. These monographs also furnish a great deal of information on generalized versions of cardinality constraints. In [10], ordinary cardinality constraints as defined above are known as *participation constraints*. In addition, [10] proposes a more general version of cardinality constraints, so-called co-occurrence constraints.

Formally, a *co-occurrence constraint* is an expression $\underline{r} : X \xrightarrow{(a,b)} Y$, where both X and Y are subsets of $\underline{r}$. A database $\underline{D}^t$ satisfies this constraint if

$$a \leq |\{z \in \pi_{X \cup Y}(\underline{r}^t) : z[X] = x\}| \leq b$$

holds for every tuple $x \in \pi_X(\underline{r}^t)$. Note that we do not claim X and Y to be disjoint, as done in [16]. It should be emphasized that co-occurrence constraints are also available in the Higher-order Entity-Relationship Model [21]. Co-occurrence constraints happen to be very powerful: for $(a, b) = (0, 1)$ they correspond to functional dependencies, for $(a, b) = (0, b)$ to numerical dependencies [11]. For results on these constraints and useful references, we refer to [13, 16, 20, 22].

Figure 2 shows a relationship type $\underline{r}$ with given participation constraints. They tell us, that every entity of type $\underline{u}$ participates in between 4 and 7 instances of $\underline{r}$, each entity of type $\underline{v}$ participates at least once, and every entity of type $\underline{w}$ at most twice. Suppose, we are given a co-occurrence constraint $\underline{r} : \{\underline{u}, \underline{w}\} \xrightarrow{(3,5)} \{\underline{v}\}$, too. Then, pairs of entities of types $\underline{u}$ and $\underline{w}$, that appear in some instance of $\underline{r}$, must co-occur (appear together) with at least 3 and at most 5 entities of type $\underline{v}$.

Example. In the university schema, cardinality constraints provide useful information: a participation constraint $card$(LECTURE, PROFESSOR) $= (3, 6)$, for example, states that every professor will give 3 to 6 lectures per term. The university of Rostock frequently offers courses for external students. Lectures for these courses are usually scheduled for weekends or during the holidays. For external students it is rather inconvenient to come to Rostock just for a single lecture. Hence, there should always be between 3 and 5 lectures per course and date. This requirement can be modeled by the co-occurrence constraint LECTURE: {COURSE, DATE} $\xrightarrow{(3,5)}$ LECTURE.

5 Pivoting under cardinality constraints

In this section, we shall discuss pivoting in the presence of cardinality constraints. To begin with, we have to define an appropriate notion of pivoting. Suppose we are given a database schema $\underline{D}$ containing a relationship type $\underline{r} = \{\underline{u}_1, \ldots, \underline{u}_n\}$.

Let S be a subset of $\underline{r}$, say $S = \{\underline{u}_1, \ldots, \underline{u}_m\}$, where $2 \leq m < n$. We define a new relationship type $\underline{s}_N$, whose component types are just the elements of S. Next, these elements shall be removed from $\underline{r}$. Therefore, we replace $\underline{r}$ by a higher-order relationship type $\underline{r}_N$ obtained from $\underline{r}$ by deleting all elements of S, and by adding the newly generated relationship type $\underline{s}_N$ as a component type. Figure 3 shows the corresponding diagrams before and after pivoting. The

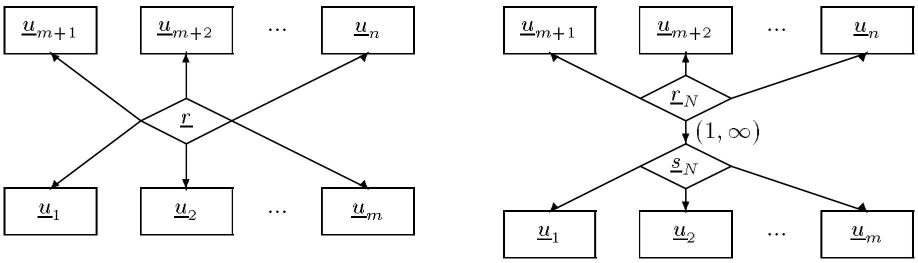

Fig. 3. ER diagrams for a database schema and its pivoted version.

elements of S are said to be the *pivoted* components of $\underline{r}$. Roughly speaking, they are shifted from $\underline{r}$ to $\underline{s}_N$.

Throughout, let $\underline{D}$ be a fixed schema, and $\underline{D}_N$ the *pivoted schema* obtained from $\underline{D}$ by adding the new relationship type $\underline{s}_N$, and replacing $\underline{r}$ by the modified relationship type $\underline{r}_N$.

Usually, we want $\underline{D}_N$ to capture the application-dependent semantics of the original schema $\underline{D}$. This natural requirement has been discussed in [2, 19, 22]. Informally, most of the proposals consider two schemas as being equivalent, if there exists a bijection between the instances of the schemas. We shall use transformation rules to show how instances of the pivoted schema are obtained from instances of the original schema:

- For each instance $r = (u_1, \ldots, u_n)$ of type $\underline{r}$, the restriction $s_N := r[S] = (u_1, \ldots, u_m)$ becomes an instance of type $\underline{s}_N$, and the tuple $r_N := (s_N, u_{m+1}, \ldots, u_n)$ becomes an instance of $\underline{r}_N$.

Vice versa, we use the following rules to (re-)transform the instances of the pivoted schema to those of the original schema:

- Let $r_N = (s_N, u_{m+1}, \ldots, u_n)$ be an instance of $\underline{r}_N$, where s_N denotes an instance $(u_1, \ldots, u_m)$ of $\underline{s}_N$. Then $r := (u_1, \ldots, u_m, u_{m+1}, \ldots, u_m)$ becomes an instance of the original relationship type $\underline{r}$.

We call two schemas *equivalent* if the transformation rules suggested above provide a bijection mapping legal instances of one schema onto legal instances of the other. Applying these transformation rules, we immediately obtain the following result.

Lemma 1 *Let Σ and Σ_N denote constraint sets defined on a schema $\underline{D}$ and its pivoted schema $\underline{D}_N$, respectively. If $(\underline{D}, \Sigma)$ and $(\underline{D}_N, \Sigma_N)$ are equivalent, then Σ_N implies the cardinality constraint $card(\underline{r}_N, \underline{s}_N) = (1, \infty)$. On the other hand, $(\underline{D}, \Sigma)$ and $(\underline{D}_N, \Sigma_N)$ are equivalent whenever $\Sigma = \emptyset$ and $\Sigma_N = \{card(\underline{r}_N, \underline{s}_N) = (1, \infty)\}$.*

Proof. Due to the transformation rules, every instance $\underline{D}^t$ is mapped one-to-one to an instance $\underline{D}_N^t$ of the pivoted schema. Herein, the instances of $\underline{s}_N$ are obtained from restrictions of instances of $\underline{r}$ to the set S of pivoted components.

Hence, every instance of $\underline{s}_N$ is involved in at least one instance of $\underline{r}_N$. This gives us the claimed participation constraint. For the same reason, this participation constraint $card(\underline{r}_N, \underline{s}_N) = (1, \infty)$ forces the transformation presented above to be onto. □

Usually, Σ is not empty, as most applications require integrity constraints. This points out a crucial problem of pivoting. All original constraints (given in Σ) have to be adjusted in the pivoted schema. Thus the question arises whether we are able to find a set of new constraints Σ_N defined on the pivoted schema, that may be used to express the original constraints. We shall consider this question in the following section. In this sense, Lemma 1 forms the basis for further investigations.

To continue with, we present a number of cases when pivoting should be taken into consideration:

- *First case:* Assume, we are given a co-occurrence constraint $\underline{r} : X \xrightarrow{(a,b)} \underline{r}$ with $b \geq 2$. Here, we suggest to choose S equal to X. In the pivoted schema, the original co-occurrence constraint may be expressed by the participation constraint $card(\underline{r}_N, \underline{s}_N) = (a, b)$.

Example. If our university schema is used to schedule the lectures for external students, we have a constraint LECTURE: {COURSE, DATE} $\xrightarrow{(3,5)}$ LECTURE. Following the proposal, we generate a new relationship type OFFER = {COURSE, DATE}, and replace the old LECTURE-type by a new one involving the types PROFESSOR, TUTOR, HALL, BUILDING as well as the newly generated type OFFER. For the sake of simplicity, we reused the old type name rather than introducing an artificial new one. In the new schema, we have the participation constraint $card$(LECTURE, OFFER) = $(3, 5)$.

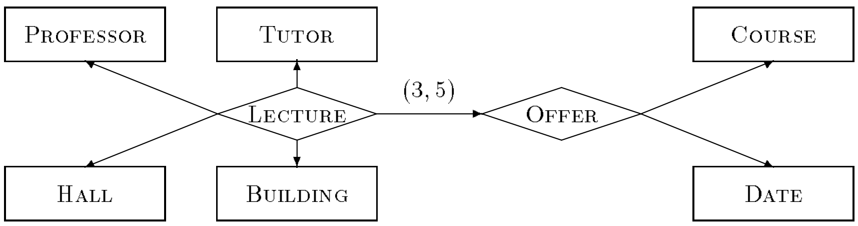

Fig. 4. The ER diagram for the pivoted schema of the university example.

In practice, it often happens that mentors have a prior engagement, or dates do not fit into the schedule of various students. Then dates have to be cancelled and replaced by new appointments. Before pivoting this information has to be updated in 3 to 5 tuples, after pivoting only in a single instance of type OFFER.

It should be mentioned, that mostly participation constraints are easier to understand and to handle than co-occurrence constraints. In particular, there exists a sound and complete system of inference rules for participation constraints, see [14, 15].

- *Second case:* Even, if we have only a co-occurrence constraint $\underline{r} : X \xrightarrow{(a,b)} Y$ with $Y \subset \underline{r}$ and $b \geq 2$, we may perform pivoting with $S = X$. This time the old co-occurrence constraint is expressed by a new one, namely $\underline{r}_N : \{\underline{s}_N\} \xrightarrow{(a,b)} Y \backslash X$.

- *Third case:* A different (and perhaps better) way to use the co-occurrence constraint $\underline{r} : X \xrightarrow{(a,b)} Y$ is the following. We choose $S = X \cup Y$, and express the old constraint by a generalized key dependency $\underline{s}_N : X \xrightarrow{(a,b)} \underline{s}_N$.

Similar to ordinary keys, these generalized key dependencies can be enforced by keeping an index and specifying this index to be such that each value appears between a and b times. Moreover, they may be used for further investigation, like horizontal decomposition as studied in [11].

Example. External students may subscribe for magazines offered by the university or its departments. Let this be modeled by the relationship type SUBSCRIPTION containing the entity types STUDENT, ADDRESS and MAGAZINE. External students usually have two addresses, a home address, and one in Rostock, where they live during their courses. Hence, we have a co-occurrence constraint SUBSCRIPTION: $\{$STUDENT$\} \xrightarrow{(1,2)} \{$ADDRESS$\}$. When pivoting as proposed above, we introduce a new relationship type SUBSCRIBER containing STUDENT and ADDRESS, and the old SUBSCRIPTION-type is replaced by a new one involving SUBSCRIBER and MAGAZINE. In the pivoted schema, we have the co-occurrence constraint SUBSCRIBER: $\{$STUDENT$\} \xrightarrow{(1,2)}$ SUBSCRIBER.

Due to our definition of pivoting in the entity-relationship context, we are now able to use both, unary and non-unary functional dependencies, for pivoting.

- *Fourth case:* If we are given a nonkey functional dependency $\underline{r} : X \to Y$, then we again choose $S = X \cup Y$. In the pivoted schema, the old functional dependency may be expressed by the key dependency $\underline{s}_N : X \to \underline{s}_N$.

If the old functional dependency is unary, i.e. X consists of a single element $\underline{u}$, then it can also be expressed by the participation constraint $card(\underline{s}_N, \underline{u}) = (0, 1)$. In data models supporting subtype or ISA-hierarchies, $\underline{s}_N$ may be considered as a subtype of $\underline{u}$. This case has been studied extensively in [5, 6].

In all four cases, pivoting removes (potential) redundancy and makes constraint enforcement more efficient. Usually, pivoting pays off only if instances of $\underline{s}_N$ are involved in more than just a single instance of $\underline{r}$. Clearly, this cannot happen if S is a key for $\underline{r}$. In this case, we do not recommend pivoting.

Before concluding this section, we have to mention that in all four cases the equivalence of the original schema and the pivoted schema can be ensured similarly to Lemma 1.

6 Preserving cardinality constraints

Due to the discussion above, the original schema and its pivoted version should be equivalent. For the designer, this requirement becomes something of a problem. We defined equivalence in terms of the instances of schemas, but these instances are not explicitly available during the design process. The designer however has to decide whether pivoting is useful or not. It goes without saying that the designer has to be supported in coping with this task.

The following theorems help us to decide whether the application-dependent constraints given in the original schema may be adjusted in the pivoted schema, or not. Our results use the idea of implication of constraints. A constraint set Σ *implies* a constraint σ if every legal database instance $\underline{D}^t$ for Σ is also legal for σ. At design time, inference rules may be used to investigate the implications of integrity constraints. Applying such rules, the designer must 'merely' focus on the purely syntactic items given in the schema.

Clearly, cardinality constraints not defined on $\underline{r}$ do not cause any problem. They may be reused in the pivoted schema, either immediately or, if $\underline{r}$ appears in the declaration of the constraints, after replacing it by $\underline{r}_N$. For cardinality constraints defined on $\underline{r}$ the problem is more involved.

Theorem 2 *Let Σ be a set of co-occurrence constraints defined on $\underline{r}$. There is a set Σ' of co-occurrence constraints such that $(\underline{D}, \Sigma)$ and $(\underline{D}_N, \Sigma_N)$ are equivalent and $\Sigma_N = \Sigma' \cup \{card(\underline{r}_N, \underline{s}_N) = (1, \infty)\}$ if and only if Σ is semantically equivalent to a constraint set consisting only of co-occurrence constraints in the form of $\underline{r} : X \xrightarrow{(a,b)} Y$ with*

$$X, Y \subseteq S, \tag{1}$$

$$X, Y \subseteq \underline{r} \backslash S, \tag{2}$$

$$S \subseteq X \text{ and } Y \subseteq \underline{r} \backslash S, \text{ or} \tag{3}$$

$$X \subseteq \underline{r} \backslash S \text{ and } S \subseteq Y. \tag{4}$$

Proof. (*Necessity.*) Without loss of generality, we may assume that every constraint in Σ satisfies one of the four conditions. Consider a co-occurrence constraint $\underline{r} : X \xrightarrow{(a,b)} Y$ in Σ. Suppose (1) holds. A database instance $\underline{D}^t$ satisfies $\underline{r} : X \xrightarrow{(a,b)} Y$ if and only if its image $\underline{D}^t_N$ satisfies $\underline{s}_N : X \xrightarrow{(a,b)} Y$. Similarly, if (2), (3) or (4) holds, then $\underline{r} : X \xrightarrow{(a,b)} Y$ is valid in $\underline{D}^t$ if and only if $\underline{D}^t_N$ satisfies $\underline{r}_N : X \xrightarrow{(a,b)} Y$, $\underline{r}_N : (X \backslash S) \cup \{\underline{s}_N\} \xrightarrow{(a,b)} Y$ or $\underline{r}_N : X \xrightarrow{(a,b)} (Y \backslash S) \cup \{\underline{s}_N\}$, respectively. This gives us the desired constraint set Σ'.

(*Sufficiency.*) Let $(\underline{D}, \Sigma)$ and $(\underline{D}_N, \Sigma_N)$ be equivalent. In the pivoted schema, we only have co-occurrence constraints declared on $\underline{r}_N$ or $\underline{s}_N$ on our disposal. An instance $\underline{D}^t_N$ of the pivoted schema satisfies $\underline{s}_N : X \xrightarrow{(a,b)} Y$ if and only if its preimage $\underline{D}^t$ satisfies $\underline{r} : X \xrightarrow{(a,b)} Y$ and (1) holds. Next, let $\underline{D}^t_N$ satisfy

$\underline{r}_N : X \xrightarrow{(a,b)} Y$. Clearly, X and Y both are subsets of $\underline{r}_N$. We distinguish four cases.

Firstly suppose, $\underline{s}_N$ is neither in X nor in Y. Then, $\underline{r}_N : X \xrightarrow{(a,b)} Y$ is valid in $\underline{D}_N^t$ if and only if $\underline{r} : X \xrightarrow{(a,b)} Y$ holds in the preimage $\underline{D}^t$. Obviously, we have (2). Secondly suppose, $\underline{s}_N$ is in X, but not in Y. Then, $\underline{r}_N : X \xrightarrow{(a,b)} Y$ holds in $\underline{D}_N^t$ if and only if the preimage $\underline{D}^t$ satisfies $\underline{r} : X \backslash \{\underline{s}_N\} \cup S \xrightarrow{(a,b)} Y$. We evidently have $S \subseteq X \backslash \{\underline{s}_N\} \cup S$ and $Y \subseteq \underline{r} \backslash S$. Thirdly, the same argument applies if $\underline{s}_N$ is in X and in Y, because an instance $\underline{D}_N^t$ satisfies either both $\underline{r}_N : X \xrightarrow{(a,b)} Y$ and $\underline{r}_N : X \xrightarrow{(a,b)} Y \backslash \{\underline{s}_N\}$, or none of them. Finally suppose, $\underline{s}_N$ is in Y, but not in X. This time, $\underline{r}_N : X \xrightarrow{(a,b)} Y$ is valid in $\underline{D}_N^t$ if and only if the preimage $\underline{D}^t$ satisfies $\underline{r} : X \xrightarrow{(a,b)} Y \backslash \{\underline{s}_N\} \cup S$. We immediately have $X \subseteq \underline{r} \backslash S$ and $S \subseteq Y \backslash \{\underline{s}_N\} \cup S$.

Consequently, a database instance $\underline{D}_N^t$ satisfies Σ_N if and only if its preimage $\underline{D}^t$ satisfies all the co-occurrence constraints derived above. Note, that each of them is in the claimed form. Since $(\underline{D}, \Sigma)$ and $(\underline{D}_N, \Sigma_N)$ are equivalent, however, the obtained constraint set is semantically equivalent to Σ as desired. □

Inference rules for co-occurrence constraints are given e.g. in [11, 13, 20, 22]. Unfortunately there does not exist a finite complete system of inference rules for co-occurrence constraints as pointed out by Grant and Minker [11]. Hence, Theorem 2 may in general be used to prove the equivalence of schemas, but not to prove their non-equivalence. However, there are subclasses of cardinality constraints that do have finite sound and complete systems of inference rules, such as the Armstrong system [1] for functional dependencies which we record here for further reference: Consider a relationship type $\underline{r}$, and subsets X, Y, Z of its component types. Then

$$\underline{r} : X \to Y \text{ holds whenever } Y \subseteq X,$$
$$\underline{r} : X \to Y \text{ and } \underline{r} : Y \to Z \text{ imply } \underline{r} : X \to Z,$$
$$\underline{r} : X \to Y \text{ implies } \underline{r} : X \cup Z \to Y \cup Z.$$

Lemma 3 *A set Σ of functional dependencies is semantically equivalent to a constraint set containing only functional dependencies $\underline{r} : X \to Y$ which satisfy one of the conditions (1–4) if and only if Σ implies one of*

$$\underline{r} : X \cap S \to S, \tag{5}$$
$$\underline{r} : X \backslash S \to S, \ \textit{or} \tag{6}$$
$$\underline{r} : X \cap S \to Y \cap S \ \textit{and} \ \underline{r} : X \backslash S \to Y \backslash S, \tag{7}$$

whenever Σ implies a functional dependency $\underline{r} : X \to Y$.

Proof. (*Sufficiency.*) We may assume, that every functional dependency in Σ satisfies one of the conditions (1–4). Consider a functional dependency

$\underline{r} : X \to Y$ in Σ. If X and Y are subsets of S, the constraints in (7) are trivially implied by Σ. The same holds when X and Y both are subsets of $\underline{r}\backslash S$. Otherwise, if X is a superset of S, the first Armstrong rule proves $\Sigma \models (5)$. Similarly, if $X \subseteq \underline{r}\backslash S$ holds and Y is a superset of S, the first and the second Armstrong rule ensure $\Sigma \models (6)$. Thus, we proved the claim for all constraints in Σ.

Every functional dependency $\underline{r} : X \to Y$ implied by Σ may be derived from Σ on applying the Armstrong rules. Consider a constraint $\underline{r} : X \to Y$ given by the first Armstrong rule. Since $Y \subseteq X$, we also have $Y \cap S \subseteq X \cap S$ and $Y\backslash S \subseteq X\backslash S$. Due to the first Armstrong rule, this gives us (7).

Next, consider a constraint $\underline{r} : X \to Z$ derived from Σ by the second Armstrong rule. Since Σ contains $\underline{r} : X \to Y$, it implies (5), (6) or (7). In the case of (5) or (6), we are ready. It remains to discuss (7). Since Σ contains $\underline{r} : Y \to Z$, too, it also implies functional dependencies similar to (5), (6) or (7), but for Y and Z. In particular, if Σ implies $\underline{r} : Y \cap S \to S$, the second Armstrong rule yields $\underline{r} : X \cap S \to S$. Analogously, if Σ implies $\underline{r} : Y\backslash S \to S$, we also get $\underline{r} : X\backslash S \to S$. If Σ implies $\underline{r} : Y \cap S \to Z \cap S$ and $\underline{r} : Y\backslash S \to Z\backslash S$, the Armstrong rules allow us to derive (7) for X and Z from Σ.

Finally, we consider a constraint $\underline{r} : X \cup Z \to Y \cup Z$ derived from Σ by the third Armstrong rule. Since $\underline{r} : X \to Y$ is in Σ, it implies (5), (6) or (7). It is easy to check, that in all three cases the third Armstrong rule may be used to derive a similar functional dependency for $X \cup Z$ and $Y \cup Z$.

(*Necessity.*) Let Σ imply (5), (6) or (7) whenever it implies $\underline{r} : X \to Y$. It suffices to construct a semantically equivalent constraint set consisting only of functional dependencies which satisfy one of the conditions in Theorem 2. Assume, there is a functional dependency $\underline{r} : X \to Y$ in Σ but not in the claimed form.

If $\Sigma \models (5)$, we replace $\underline{r} : X \to Y$ by $\underline{r} : X \cup S \to Y\backslash S$ and $\underline{r} : X \cap S \to S$. If $\Sigma \models (6)$, we replace $\underline{r} : X \to Y$ by $\underline{r} : X \cup S \to Y\backslash S$ and $\underline{r} : X\backslash S \to S$. Finally, if $\Sigma \models (7)$, we replace $\underline{r} : X \to Y$ by $\underline{r} : X \cap S \to Y \cap S$ and $\underline{r} : X\backslash S \to Y\backslash S$.

In all three cases, Σ and the resultant constraint set Σ' are semantically equivalent. Moreover, Σ' contains less functional dependencies not in the desired form than Σ does. Since Σ is finite, the repeated application of the suggested modification will finally yield a semantically equivalent constraint set without any functional dependencies not in the desired form. This concludes the proof. □

Closure operations are a useful and interesting instrument for investigating functional dependencies, see [1]. Given a set Σ of functional dependencies defined on $\underline{r}$, and any subset $X \subseteq \underline{r}$, let $cl_\Sigma(X) = \{\underline{u} \in \underline{r} : \Sigma \text{ implies } (\underline{r} : X \to \{\underline{u}\})\}$ denote the closure of X under functional dependencies.

Theorem 4 *Let Σ be a set of functional dependencies defined on $\underline{r}$. Then there is a set Σ' of functional dependencies such that $(\underline{D}, \Sigma)$ and $(\underline{D}_N, \Sigma_N)$ are equivalent and $\Sigma_N = \Sigma' \cup \{card(\underline{r}_N, \underline{s}_N) = (1, \infty)\}$ if and only if for every set $X \subseteq \underline{r}$*

one of the following conditions hold:

$$S \subseteq cl_\Sigma(X \cap S), \tag{8}$$
$$S \subseteq cl_\Sigma(X \backslash S), \text{ or} \tag{9}$$
$$cl_\Sigma(X) = cl_\Sigma(X \cap S) \cup cl_\Sigma(X \backslash S), \tag{10}$$
$$\text{where } cl_\Sigma(X \cap S) \subseteq S \cup cl_\Sigma(\emptyset) \text{ and } cl_\Sigma(X \backslash S) \subseteq (\underline{r} \backslash S) \cup cl_\Sigma(\emptyset).$$

Proof. (*Sufficiency.*) It is easy to check, that Theorem 2 remains true if we restrict ourselves to functional dependencies, i.e. to pairs $(a, b) = (0, 1)$. Consequently, we may apply Lemma 3 to Σ. Consider a set $X \subseteq \underline{r}$. To begin with, let X be a subset of S. Σ implies $\underline{r} : X \to cl_\Sigma(X)$, and by virtue of Lemma 3, Σ also implies $cl_\Sigma(X) \subseteq S \cup cl_\Sigma(\emptyset)$ or $S \subseteq cl_\Sigma(X)$. Similarly, we derive $cl_\Sigma(X) \subseteq (\underline{r} \backslash S) \cup cl_\Sigma(\emptyset)$ or $S \subseteq cl_\Sigma(X)$ when X is a subset of $\underline{r} \backslash S$.

Now, let X be an arbitrary subset of $\underline{r}$. Again we use Lemma 3. If Σ implies $\underline{r} : X \cap S \to S$, we immediately derive $S \subseteq cl_\Sigma(X \cap S)$. If Σ implies $\underline{r} : X \backslash S \to S$, we clearly obtain $S \subseteq cl_\Sigma(X \backslash S)$.

Conversely, assume Σ implies neither $\underline{r} : X \cap S \to S$ nor $\underline{r} : X \backslash S \to S$. As pointed out above, we may conclude $cl_\Sigma(X \cap S) \subseteq S \cup cl_\Sigma(\emptyset)$ as well as $cl_\Sigma(X \backslash S) \subseteq (\underline{r} \backslash S) \cup cl_\Sigma(\emptyset)$. Moreover, Σ implies $\underline{r} : X \cap S \to cl_\Sigma(X) \cap S$ and $\underline{r} : X \backslash S \to cl_\Sigma(X) \backslash S$ due to Lemma 3. This gives us $cl_\Sigma(X) \cap S \subseteq cl_\Sigma(X \cap S)$ and $cl_\Sigma(X) \backslash S \subseteq cl_\Sigma(X \backslash S)$. Together, these observations yield $cl_\Sigma(X) = cl_\Sigma(X \cap S) \cup cl_\Sigma(X \backslash S)$, i.e. (10) as claimed.

(*Necessity.*) As shown in [1], Σ is semantically equivalent to the constraint set containing the functional dependencies $\underline{r} : X \to cl_\Sigma(X)$ for all subsets $X \subseteq \underline{r}$. Obviously, we may apply Lemma 3 to Σ. Furthermore, if we use the analogue of Theorem 2 for functional dependencies as suggested above, we obtain the desired constraint set Σ_N. □

Theorem 4 in particular implies Theorem 6 of [5], which applies for pivoting guided by a unary functional dependency $\underline{r} : \{\underline{u}\} \to S$.

Finally, we give a similar result for participation constraints. A sound and complete system of inference rules is available in [14, 15].

Theorem 5 *Let Σ be a set of participation constraints defined on $\underline{r}$. Then there is a set Σ_N of participation constraints such that $(\underline{D}, \Sigma)$ and $(\underline{D}_N, \Sigma_N)$ are equivalent if and only if every participation constraint $card(\underline{r}, \underline{u}) = (a, b)$ implied by Σ satisfies one of the following conditions:*

$$\underline{u} \in \underline{r} \backslash S, \tag{11}$$
$$\underline{u} \in S, \text{ and } a \leq 1 \text{ and } b = \infty, \tag{12}$$

or there is some component type $\underline{w} \in S$ such that Σ implies $card(\underline{r}, \underline{w}) = (0, 1)$ or $(1, 1)$.

Proof. It is noteworthy that Σ implies the cardinality constraint $card(\underline{r}, \underline{w}) = (0,1)$ or $(1,1)$ for some pivoted component type $\underline{w}$ if and only if Σ implies the key dependency $\underline{r} : S \to \underline{r}$, see [15].

(*Sufficiency.*) Let $(\underline{D}, \Sigma)$ and $(\underline{D}_N, \Sigma_N)$ be equivalent. Consider a pivoted component type $\underline{u} \in S$. Assume, $card(\underline{r}, \underline{u}) = (a, b)$ is the sharpest cardinality constraint implied by Σ for the link $(\underline{r}, \underline{u})$. Analogously, let $card(\underline{s}_N, \underline{u}) = (a_N, b_N)$ and $card(\underline{r}_N, \underline{s}_N) = (a_N^S, b_N^S)$ be the sharpest cardinality constraints implied by Σ_N for the links $(\underline{s}_N, \underline{u})$ and $(\underline{r}_N, \underline{s}_N)$, respectively.

Let $a \leq d \leq b$. Due to [14], there is a legal database $\underline{D}^t$ containing an instance $u \in \underline{u}^t$ that participates in exactly d instances of $\underline{r}$. As pointed out in [15], we may construct $\underline{D}^t$ such that the instances of $\underline{r}$ mutually agree in at most one entry. Since $|S| \geq 2$, the restrictions $r[S]$ are mutually distinct for $r \in \underline{r}^t$. Hence, in the image $\underline{D}_N^t$ every instance of $\underline{s}_N$ participates in exactly one instance of $\underline{r}_N$. This gives us $a_N^S = 1$, $a_N \leq a$ and $b \leq b_N$. On the other hand, if $d \geq 2$ and S is not a key for $\underline{r}$, it is not d7icult to construct $\underline{D}^t$ such that u participates in at least two instances r_1 and r_2 of $\underline{r}$ with $r_1[S] = r_2[S]$. In this case, u will participate in less than d instances of $\underline{s}_N$ in the image $\underline{D}_N^t$. This yields $a_N < a$ when $a \geq 2$ and S is not a key.

Now, let $a_N \leq d_N \leq b_N$ and $1 = a_N^S \leq d_N^S \leq b_N^S$. Recall, that there is a legal database $\underline{D}_N^t$ containing an instance $u \in \underline{u}^t$ that participates in exactly d_N instances of $\underline{s}_N$ such that one of these instances occurs in d_N^S instances of $\underline{r}_N$. Therefore, in the preimage $\underline{D}^t$, this u participates in $d_N d_N^S$ instances of $\underline{r}$. We obtain $a \leq a_N a_N^S = a_N$ and $b_N b_N^S \leq b$.

Combining both observations, we find that $a_N = a$ and $b_N = b$ together with $b_N^S = 1$ or $b = \infty$. In the first case, S is clearly a key for $\underline{r}$. Conversely, if S is not a key, we additionally derive $a < 2$, which gives us (12).

(*Necessity.*) It remains to construct a suitable constraint set Σ_N. Let $card(\underline{r}, u) = (a, b)$ be in Σ. If $\underline{u} \notin S$, we put $card(\underline{r}_N, u) = (a, b)$ into Σ_N. Conversely, if $\underline{u} \in S$, we add $card(\underline{s}_N, u) = (a, b)$ to Σ_N. Moreover, let Σ_N contain $card(\underline{r}_N, \underline{s}_N) = (1, 1)$ if S is a key for $\underline{r}$, or $card(\underline{r}_N, \underline{s}_N) = (1, \infty)$ otherwise. It is easy to check, that Σ_N is the desired set of cardinality constraints. □

7 Path cardinality constraints

Due to our definition of pivoting, this transformation is always lossless, but not necessarily constraint-preserving. Theorem 5, for example, states that most participation constraints declared for component types in S cannot be adjusted, unless there is a type $\underline{w} \in S$, which forms a key for the original relationship type $\underline{r}$. But in this case, as argued before, pivoting should not be applied. The problem, in general, is that we do not know exactly (neither in theory, nor in practice) how often instances of $\underline{s}_N$ are involved in instances of type $\underline{r}_N$.

Example. Remember our university schema. Motivated by the co-occurrence constraint LECTURE: {COURSE, DATE} $\xrightarrow{(3,5)}$ LECTURE, we performed pivoting and obtained new relationship types OFFER and LECTURE together with the participation constraint $card$(LECTURE, OFFER) $= (3,5)$.

Now assume, the university decides to organize between 6 and 12 lectures per course. In the original schema, this corresponds to the participation constraint $card$(LECTURE, COURSE) $= (6,12)$. How can we adjust this condition in the pivoted schema? It is easy to check, that $card$(OFFER, COURSE) $= (2,3)$ is too sharp, while $card$(OFFER, COURSE) $= (2,4)$ is not sharp enough.

In order to cope with this problem, we have to strengthen the concept of cardinality constraint. A (directed) *path* $P = (\underline{v}_0, \underline{v}_1, \ldots, \underline{v}_k)$ is a sequence of links $(\underline{v}_0, \underline{v}_1), (\underline{v}_1, \underline{v}_2), \ldots, (\underline{v}_{k-1}, \underline{v}_k)$ in the entity-relationship diagram of a schema. Here, $\underline{v}_0$ is the initial vertex of the path, and $\underline{v}_k$ is its terminal vertex. The integer k denotes the length of the path. Thus, paths of length 1 are just links. For standard graph-theoretic terminology, we refer to [12].

So far, when discussing cardinality constraints, we only considered component types of $\underline{r}$, i.e. types connected to $\underline{r}$ via a link. Now, we shall extend this concept to all types reachable from $\underline{r}$ via a path in the entity-relationship diagram. In the object-oriented context, this idea is well-known and formalized by path functions.

A *path function* ϕ_P maps instances of the initial vertex to instances of the terminal vertex. Path functions can be defined recursively as follows: To begin with, let $L = (\underline{r}, \underline{u})$ be a link. For every instance r of $\underline{r}$, we put $\phi_L(r) := u$ where u is the restriction of r to the component type $\underline{u}$. Next, consider any path $P = (\underline{r}, \underline{v}_1, \ldots, \underline{v}_{k-1}, \underline{v}_k)$ of length k. Let $Q = (\underline{r}, \underline{v}_1, \ldots, \underline{v}_{k-1})$ be its subpath of length $k-1$, and $L = (\underline{v}_{k-1}, \underline{v}_k)$ be the remaining link. Then we put $\phi_P(r) := \phi_L(\phi_Q(r))$ for an instance r of type $\underline{r}$.

Now, we are ready to redefine the concept of cardinality constraints. Consider a path P with initial vertex $\underline{r}$ and terminal vertex $\underline{u}$. A *path participation constraint* is an expression $card\ P = (a,b)$. A database instance $\underline{D}^t$ satisfies this constraint if for every instance $u \in \underline{u}^t$ there are at least a and at most b instances $r \in \underline{r}^t$ such that $\phi_P(r) = u$ holds. If the path consists of a single link, this definition recaptures the original one in Section 4.

Example. In our university example, the pivoted schema contains a path P=(LECTURE, OFFER, COURSE). Using this path, the participation constraint $card$(LECTURE, COURSE) $= (6,12)$ in the original schema may be replaced by the path participation constraint $card$(LECTURE, OFFER, COURSE) $= (6,12)$ in the pivoted schema.

In a similar way, we proceed for co-occurrence constraints: While ordinary co-occurrence constraints relate some of the components of tuples r in a population $\underline{r}^t$, path co-occurrence constraints relate some of instances reachable by path functions from tuples $r \in \underline{r}^t$.

Let $X = \{P_1, \dots, P_l\}$ be a set of paths with common initial vertex $\underline{r}$, and terminal vertices $\underline{v}_1, \dots, \underline{v}_l$, respectively. For an instance r of type $\underline{r}$, we define $r[X] := (v_1, \dots, v_l) \in \underline{v}_1^t \times \dots \times \underline{v}_l^t$ by claiming $\phi_{P_i}(r) = v_i$, for $i = 1, \dots, l$.

A *path co-occurrence constraint* is an expression $\underline{r} : X \xrightarrow{(a,b)} Y$, where both, X and Y are sets of paths with common initial vertex $\underline{r}$. A database instance $\underline{D}^t$ satisfies this constraint if

$$a \leq |\{r[X \cup Y] : r \in \underline{r}^t \text{ and } r[X] = (v_1, \dots, v_l)\}| \leq b$$

holds for every $(v_1, \dots, v_l) \in \underline{v}_1^t \times \dots \times \underline{v}_l^t$, or there does not exist any $r \in \underline{r}^t$ satisfying $r[X] = (v_1, \dots, v_l)$ at all.

Again, path co-occurrence constraints of the form $\underline{r} : X \xrightarrow{(0,1)} Y$ correspond to path functional dependencies, which have been first studied by Weddell [24], and in a slightly stronger setting by Thalheim [21]. In both papers, you will also find a sound and complete system of inference rules for path functional dependencies.

In the sequel, we use the notion of a *path cardinality constraint* if we refer to a path participation or path co-occurrence constraint. Theorem 6 proves pivoting to be constraint-preserving, due to the expressive power of path cardinality constraints. In the special case, where pivoting is guided by a unary functional dependency, a similar result for path functional dependencies is given in [6].

Theorem 6 *Let Σ be a set of path cardinality constraints defined on $\underline{D}$. Then there is a set Σ_N of path cardinality constraints such that $(\underline{D}, \Sigma)$ and $(\underline{D}_N, \Sigma_N)$ are equivalent.*

Proof. The theorem may be verified using the transformation rules presented in Section 5. Consider a path P of the original schema used in a constraint in Σ. If P does not contain the fixed relationship type $\underline{r}$, then P is also a path of the pivoted schema. Otherwise, P has to be replaced by a modified path P_N. If P contains a link $(\underline{r}, \underline{u})$ with $\underline{u} \in S$, we replace this link by the new links $(\underline{r}_N, \underline{s}_N)$ and $(\underline{s}_N, \underline{u})$ to obtain P_N. If not, we simply replace $\underline{r}$ by $\underline{r}_N$ to get P_N. Clearly, P_N is a path of the pivoted schema. Now, for every constraint in Σ using P in its declaration, we replace P by P_N. As a result, we obtain Σ_N. Due to Lemma 1, it remains to add the (path) participation constraint $card(\underline{r}_N, \underline{s}_N) = (1, \infty)$. □

Finally, a brief remark on the path manipulations is called for. As seen above, pivoting might increase the length of paths used in a cardinality constraint by 1. In a path co-occurrence constraint $\underline{r}_N : X \xrightarrow{(a,b)} Y$ where all paths in X and in Y start with the link $(\underline{r}_N, \underline{s}_N)$, we may delete the initial vertex $\underline{r}_N$ from each path, and redeclare this constraint on the relationship type $\underline{s}_N$. In practice, this often happens. The simplification has been applied in Theorems 2 and 4. For participation constraints however, Theorem 5 states that this simplification is very rare.

8 Concluding remarks

Pivoting was suggested in [5] for breaking-up a relationship type on the semantic level. It generalizes the well-known decomposition of a relation type guided by a functional dependency. We continued this study by investigating pivoting in the presence of cardinality constraints. In Section 5, we defined a suitable notion of pivoting in an extended entity-relationship model. This is of interest as the entity-relationship model is not only popular as an intuitive tool for conceptual modeling, but becomes more and more prevalent as a basis upon which real database management systems are built.

It is natural to ask, whether all application-dependent constraints may be adjusted in the pivoted schema. Examples served to illustrate that this is not always possible, at least not if we insist on the usual concept of cardinality constraint. To overcome this limitation, we defined path cardinality constraints and proved that this generalization is sufficient to ensure the equivalence of the original schema and its pivoted version.

Our discussion showed that pivoting does not necessarily result in a schema which is 'better' than the original one. On the one side, pivoting may be used to increase the efficiency of constraint enforcement by reducing redundancy. On the other side, we have to cope with paths in the constraints. The decision depends on the pay-off expected from decomposition during the life-time of the designed database.

In Section 5, we assembled a number of cases when pivoting is reasonable. They should be considered as rules of thumb that can assist a designer in the design process. In Section 6, we characterized the well-behaving cases when path constraints do not come into play.

We believe that, path participation constraints shall not be seen as an obstacle that prevents a designer from pivoting a relationship type whenever this is promising. These constraints are easy to understand, and to enforce in a database system. It would be interesting to have a complete set of inference rules for path participation constraints. Unfortunately, reasoning about these constraints turns out to be different from the original case. While for ordinary participation constraints, the consistency merely depends on the existence of critical cycles [17] in the entity-relationship diagram, this does hold not for path participation constraints. However this issue is out of the scope of the paper, but suggested for future investigation.

References

1. W.W. Armstrong, Dependency structures of database relationship, Information Processing 74 (1974) 580-583.
2. P. Assenova and P. Johannesson, Improving quality in conceptual modelling by the use of schema transformations, in: B. Thalheim (ed.), Conceptual Modeling (Springer, Berlin, 1996) 277-291.
3. C. Batini, S. Ceri and S.B. Navathe, Database design with the ER model, (Benjamin/Cummings, Menlo Park, 1991).

4. J. Biskup, R. Menzel and T. Polle, Transforming an entity-relationship schema into object-oriented database schemas, in: Proc. ADBIS'95 (Moscow, 1995) 67-78.
5. J. Biskup, R. Menzel, T. Polle and Y. Sagiv, Decomposition of relationships through pivoting, in: B. Thalheim (ed.), Conceptual Modeling (Springer, Berlin, 1996) 28-41.
6. J. Biskup and T. Polle, Decomposition of database classes under path functional dependencies and onto constraints, in: B. Thalheim and K.-D. Schewe (eds.), Foundations of Information and Knowledge systems (Springer, Berlin, 2000) 31-49.
7. D. Calvanese and M. Lenzerini, Making object-oriented schemas more expressive, Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, (ACM Press, Minneapolis, 1994) 243-254.
8. P.P. Chen, The Entity-Relationship Model: Towards a unified view of data, ACM Trans. Database Syst. 1 (1976) 9-36.
9. E.F. Codd, A relation model of data for large shared data banks, Commun. ACM 13 (1970) 377-387.
10. D.W. Embley, B.D. Kurtz and S.N. Woodfield, Object oriented systems analysis: a model-driven approach, (Yourdon Press Series, Prentice Hall, 1992).
11. J. Grant and J. Minker, Inferences for numerical dependencies, Theoretical Comput. Sci. 41 (1985) 271-287.
12. J. Gross and J. Yellen, Graph theory, (CRC press, Boca Raton, 1999).
13. S. Hartmann, Über die Charakterisierung und Konstruktion von ER-Datenbanken mit Kardinalitätsbedingungen, Ph.D. thesis (Rostock, 1996).
14. S. Hartmann, On the consistency of int-cardinality constraints, in: T.W. Ling, S. Ram and M.L. Lee (eds.), Conceptual Modeling, LNCS 1507 (Springer, Berlin, 1998) 150-163.
15. S. Hartmann, On interactions of cardinality constraints, key and functional dependencies, in: B. Thalheim and K.-D. Schewe (eds.), Foundations of Information and Knowledge systems (Springer, Berlin, 2000) 31-49.
16. S.W. Liddle, D.W. Embley and S.N. Woodfield, Cardinality constraints in semantic data models, Data Knowl. Eng. 11 (1993) 235-270.
17. M. Lenzerini and P. Nobili, On the satisfiability of dependency constraints in Entity-Relationship schemata, Information Systems 15 (1990) 453-461.
18. H. Mannila and K. Räihä, The design of relational databases, (Addison-Wesley, Reading, 1992).
19. J.A. Makowsky and E.V. Ravve, Dependency preserving refinements and the fundamental problem of database design, Data Knowl. Eng. 24 (1998) 277-312.
20. A. McAllister, Complete rules for n-ary relationship cardinality constraints, Data Knowl. Eng. 27 (1998) 255-288.
21. B. Thalheim, Foundations of Entity-Relationship Modeling, Ann. Math. Artif. Intell. 6 (1992) 197-256.
22. B. Thalheim, Entity-relationship modeling (Springer, Berlin, 2000).
23. J.D. Ullman, Principles of database and knowledge-base systems, Vol. I (Computer Science Press, Rockville, 1988).
24. G.E. Weddell, Reasoning about functional dependencies generalized for semantic data models, ACM Trans. Database Syst. 17 (1992) 32-64.