Serdica J. Computing **3** (2009), 179–204

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

REASONING METHODS FOR DESIGNING AND SURVEYING RELATIONSHIPS DESCRIBED BY SETS OF FUNCTIONAL CONSTRAINTS*

János Demetrovics, András Molnár, Bernhard Thalheim

Abstract. Current methods of database schema design are usually based on modeling the real world as entity (or object) classes with relationships among them. Properties of relationships can be described by semantical database constraints. One of them is functional dependency, which has a key role in traditional database design. The three basic types of binary relationships that can be described by functional dependencies are one-to-one, one-to-many and many-to-many. They can also be expressed by common graphical languages like the Entity-Relationship (ER) graph. However, relationships defined among more than two entity classes (ternary, quaternary, etc.) are usually not investigated and the common graphical tools lack expressive power regarding them. We show that the variety of relationship types is rich for higher arities and propose a simplified formalism for functional constraints as well as graphical and spreadsheet reasoning methods for handling sets of functional constraints that also help by relationship design.

ACM Computing Classification System (1998): E.1.

Key words: Database schema, functional dependency, Entity-Relationship model, functional constraint, reasoning method.

*This work is supported by the Hungarian Scientific Research Fund (OTKA), grant T042706.

1. Introduction

1.1. Relations, relationships and constraints. In relational database theory, the syntactical part of a *database schema* consists of finitely many *relational schemata*, and each relation schema has one or more (finitely many) *attributes*. It describes the structure of the data. The actual data contained in a *database instance* consist of *relational tables* for each relation schema such that the attributes correspond to columns of the table. The schema is extended by semantical *integrity constraints* to ensure consistency of the database by specifying which instances are considered as valid databases. The database management system checks that the prescribed constraints are not violated by any transaction.

Functional dependencies are probably the most known database constraints. For two sets of attributes X, Y of a relational schema R, $X \to Y$ states that only those tables are treated as valid instances of R that contain no pair of rows which have the same values in the columns of attributes X but differ in any of the columns of attributes Y, i. e., the values of X uniquely determine the values of Y. For instance, in an address table of schema $(City, ZIP, Street, HouseNr)$, the ZIP code determines $City$ so there is a functional dependency $ZIP \to City$ between them. A *key* is a set of attributes that functionally determines all attributes of the relation schema. It is an important task during schema design to determine which of the possible functional dependencies are valid, in order to avoid update anomalies and find a suitable decomposition of the relational schemata. Specified constraints are not independent of each other: one or more constraints may logically *imply* other constraints while some possible constraints remain independent. An *axiomatization* of functional constraints consists of axioms and rules that support reasoning on constraints.

The notion of functional dependencies was introduced in [2] for the relational database model [13], mainly to provide a way for specification of the properties of valid, acceptable instances of a relational schema. Since then, the theory of functional dependencies has been well developed by investigating properties and representations, applied for key finding, decompositions, normal forms, etc. Classical database design is based on a step-wise extension of the constraint set and on a consideration of constraint sets through generation by tools. Nowadays, dependency theory at schema design is usually applied for determining keys and decomposing schemata into normal forms (e. g., [5, 6, 4, 7]).

The *Entity-Relationship (ER) model* (e. g., [11, 12, 10, 24]) is the most widely used graphical tool for database schema design. The design procedure is based on identification of entity classes and relationships among them. Entity classes are represented as rectangles and relationship classes as diamonds

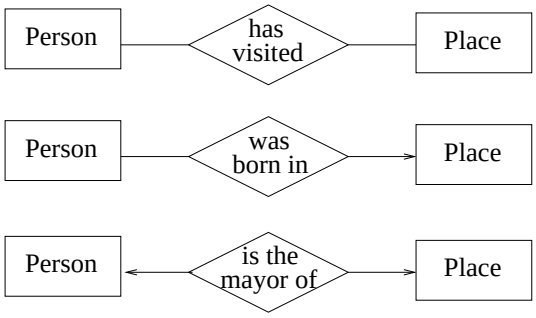

Fig. 1. Three basic types of binary relationships in the Entity-Relationship model (arrowheads representing cardinality restrictions similarly to functional dependencies)

connected to the participating entity classes. Relationships are usually binary, ie. two entity classes participate in the relationship. Relationships of higher arity (e. g., ternary, quaternary) are those that have more than two entity classes participating in them. They are also allowed and should be used whenever convenient.

The model allows specification of cardinalities of entities participating in relationships. Since we are focusing on relationships that can be described by sets of functional dependencies, the only possible cardinalities we consider are *'at most one'*, usually denoted by $(0, 1)$, and *'any'* $(0, n)$. It refers to how many times an entity can participate in a particular relationship (whether it can relate to more than one entity of the other class or not). A simplified notation is when an arrowhead is used on the opposite side to represent a $(0, 1)$ cardinality. It harmonizes with the functional dependency notation: an entity that participates only once uniquely determines the entity it is related to via the relationship.

The three basic types of binary relationships that can also be expressed by functional constraints are *many-to-many, many-to-one and one-to-one*. Figure 1 shows an example for each type. In the first relationship, any person can have visited any number of places (villages/towns/cities) and a place can have been visited by an unrestricted number of people. It is a many-to-many relationship that can be specified by the empty set of functional dependencies. In the second case, a person was born at exactly one place but several persons may have been born at the same place and so it is a many-to-one relationship corresponding to the singleton functional dependency set $\{Person \rightarrow Place\}$. In the third case, we assume a place has currently one mayor and a person can be the mayor of only one place so there is a one-to-one relationship between them: $\{Person \rightarrow Place, Place \rightarrow Person\}$. Note that if we allowed a person to be the mayor of

more than one place at the same time, we would get a *one-to-many* relationship as a fourth case. However, it is basically of the same type as many-to-one if we change the roles of the two participating entity classes.

A relational database schema from an ER graph can be generated automatically: entity and relationship classes are transformed to relational schemata with attributes and integrity constraints. If relationship types are specified exactly in the above terms, this transformation process can optimize the generated schema by eliminating redundancies (schema normalization).

The general entity-relationship model allows more sophisticated specification of cardinalities, such as optionality and exact numbers instead of the notation n. In the above examples, one may specify that a place must have a mayor. However, *inclusion dependencies* would be needed in addition to functional dependencies to describe that an entity must participate in a relationship (e. g., for a $(1, n)$ or $(1, 1)$ cardinality). Therefore, optionality is not considered by us as an aspect of relationship type. Furthermore, the general entity-relationship language allows specification of cardinalities like $(2, 3)$ (at least 2, at most 3) that are not captured by functional constraints. We will show in this paper that the variety of relationship types is already rich for higher arities if we consider only the simple case of relationships described by sets of functional dependencies.

Consider the two examples on Figure 2. In the first case, each person should have one birth date and place. In the second case, however, each person can have moved several times but only once on a day and only once to a specific

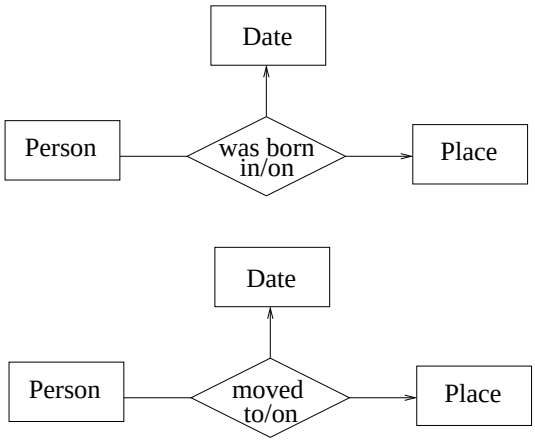

Fig. 2. Ambiguity of ternary relationship specification in the Entity-Relationship model

place (the example assumes nobody can move back to a previous place—or if so, previous dates of moves to the same place are not stored). It is clear that the structure of the two ternary relationships differs essentially, which is not captured by the entity–relationship notation and other common graphical tools[1]. The first case has the functional dependency set $\{Person \to Date, Person \to Place\}$ while the second has $\{PersonPlace \to Date, PersonDate \to Place\}$. In a later phase of schema design, they can be decomposed to binary relationships. However, the *binarization* of relationships usually needs introducing extra relationship or even entity classes (e. g., as in the second case of our example). While binarization can be useful in some cases, it generally complicates the graph notation and can make the schema less natural and perspicuous already at a relatively early stage of the database design process.

Since designing a relationship of higher arity can be incomplete or ambiguous, and some languages don't even allow specification of non-binary relationships, binarization is often performed even if higher arity relationships would provide a more suitable model. In fact, the complexity of this kind of relationships can be high and different types of ternary, quaternary relationships are not characterized (as opposed to the well-known three types of binary cases). Complete and unambigous specification can be achieved by recalling database constraints, and relationship construction can be achieved through relation schema design. To achieve this, the database developer must master acquisition of semantics. The traditional formal notation considers dependencies one-by-one, including trivial and redundant ones. The implication is not effective enough in most cases. There is usually a strong inter-dependence among constraints that is not visualized. All these lead to inconvenience in using the formalism with the traditional axiomatization. Therefore, simple and sophisticated means of representation and reasoning for constraint sets are needed.

In [20] we proposed a novel approach for graphical representation of sets of functional dependencies for small relation schemata that supports reasoning. In this paper we give extended methods for graphical reasoning using this representation, and introduce an alternative, spreadsheet-based representation of constraint sets. More details on the graphical and spreadsheet representations can be found in our technical report [19].

The proposal of a graphical representation of constraint sets provides a possible solution of the problem of defining a pragmatical approach that allows

[1]Instead of the arrows we can use different versions of the original cardinality constraint notation as participation or *look-through* constraints [24]. However, neither of them is capable to describe precisely all the possible ternary relationship structures.

simple representation of and reasoning on database constraints. The constraint acquisition method below can be refined and adopted into this framework. Solving the problem is crucial since typical algorithms such as normalization algorithms can only generate a correct result if the specification is complete, i. e., each possible constraint is declared as either valid or invalid in a consistent, non-contradictory way.

1.2. Functional constraints, excluded functional constraints and their dimension. We use the traditional functional dependency notation with some restrictions. Beside functional dependencies (FDs), we use *excluded functional constraints* (also called *negated functional dependencies*) as well: e. g., $X \not\longrightarrow Y$ states that the functional dependency $X \longrightarrow Y$ is not valid[2].

In our notation, a *trivial (redundant)* constraint (a functional dependency or an excluded functional constraint) is a constraint with at least one attribute of its left-hand side and right-hand side in common or with the empty set as its right-hand side. Furthermore, a *canonical (singleton)* functional dependency or a singleton excluded functional constraint has exactly one attribute on its right-hand side.

Our graphical and spreadsheet representations deal with non-trivial canonical functional dependencies and non-trivial singleton excluded functional constraints only. Restriction means we explicitly disallow trivial (redundant) and non-singleton constraints and treat them as syntactically invalid. We will show in Section 3 (see also [19]) that this leads to simple and effective representation of constraint sets.

Some constraints logically imply others. If a constraint $X \to A$ holds in each possible schema where a set of constraints $\mathcal{F}$ holds, this is denoted by $\mathcal{F} \vDash X \to A$. Implication can be checked given an axiomatization. If an implication can be derived formally using an axiomatization $\mathcal{A}$, this is denoted by $\mathcal{F} \vdash_{\mathcal{A}} X \to A$. $\mathcal{A}$ is *sound* if each derivation is correct ($\vDash$ follows from $\vdash_{\mathcal{A}}$) and is *complete* if each logical implication can be derived using $\mathcal{A}$ (ie. $\vDash$ implies $\vdash_{\mathcal{A}}$). The notation is extended to negated constraints in a straightforward way. We will show that restriction to non-trivial and singleton constraints can be achieved without losing relevant deductive power.

In most of the cases, we focus on *closed* sets of functional dependencies. A finite (singleton, non-trivial) constraint set $\mathcal{F}$ is closed iff $\mathcal{F}^+ = \mathcal{F}$ where $\mathcal{F}^+$

[2]Negated functional dependencies are interpreted by *weak semantics*, i. e., a negated dependency is allowed to be valid in certain instances but this is not the general case for instances of the schema.

is the (singleton, non-trivial) closure of $\mathcal{F}$, i. e., contains all implied singleton, non-trivial constraints.

Dimension of a constraint is simply the size of its left-hand side, i. e., the number of attributes on its left-hand side.

For a single attribute A, given a set of functional dependencies $\mathcal{F} \subset \mathbb{D}_c^+$, the *dimension of* A is denoted by $[A]_{\mathcal{F}}$ (or simply $[A]$) and defined as

$$[A]_{\mathcal{F}} \stackrel{\text{def}}{=} \min_{X \to A \in \mathcal{F}^+} |X|$$

This definition is extended with $[A]_{\mathcal{F}} \stackrel{\text{def}}{=} \infty$ for the case when no $X \to A$ exists in $\mathcal{F}^+$. The dimensions of attributes classify the sets of functional dependencies.

1.3. Constraint set development. The main task is to determine the validity of all possible functional dependencies given an initial set, i. e., to get the closure of the constraint set. This constraint acquisition is usually performed by a step-wise extension of the constraint set. The approach is based on the separation of constraints into:

The set of valid functional dependencies Σ_1**:** All dependencies that are known to be valid and all those that can be implied from the set of valid and excluded functional dependencies.

The set of excluded functional dependencies Σ_0**:** All dependencies that are known to be invalid and all those that are invalid and can be implied from the set of valid and excluded functional dependencies.

This pragmatical approach leads to the following simple elicitation algorithm illustrated by Figure 3.

1. *Basic step: Design obvious constraints.*
2. *Recursion step: Repeat until the constraint sets* Σ_0 *and* Σ_1 *do not change.*

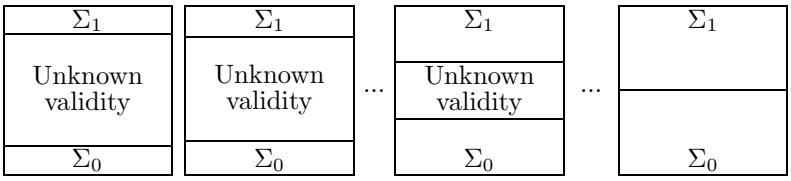

Fig. 3. Constraint Acquisition Process

- *Find a functional dependency α that is neither in Σ_1 nor in Σ_0. If α is valid then add α to Σ_1. If α is invalid then add α to Σ_0.*
- *Generate the logical closures of Σ_0 and Σ_1.*

The number of constraints may however be exponential in the number of attributes [17]. Therefore the specification of the complete set of functional dependencies may be a task that is infeasible. This problem is closely related to another well-known combinatoric problem presented during MFDBS'87 [23], still only partially solved: What is the size of sets of independent functional dependencies for an n-ary relation schema?

The above constraint acquisition process however turns out to be too static in some cases. The tasks of a designer may include generating closures of attribute sets or determining whether a specific functional dependency holds or not, even before generating the closure. Some of these methods are adaptations of well-known algorithms for our representation systems, while others are based on an implication system especially designed for simple handling of constraints. These methods also point towards an interactive software tool to be implemented for supporting design of schemata (relationships) as well as surveying types of possible schemata.

The outline of the paper is as follows: Section 2 introduces our proposal for spreadsheet and graphical representations of functional constraint sets and also discusses the number of different sets. Section 3 gives a simple axiomatization for singleton, nontrivial constraints, demonstrates the usage of its rules in terms of the graphical and spreadsheet representations for reasoning on constraint sets, and finally sketches some auxilliary methods that can take further advantage of our representations during database schema design.

2. Sets of functional dependencies and their representations.

2.1. The spreadsheet notation of sets of functional dependencies.

A set of functional dependencies over a specific set of attributes can be represented as a row of a table where columns correspond to the possible functional dependencies and a digit 1 or 0 in a column indicates the presence or absence of the corresponding dependency in the set. This representation is a brief but still convenient way to present a larger amount of sets and can also be used for reasoning on a particular set of constraints.

Since our main focus is on schema design and relationship types, we ignore cases with zero-dimensional constraints (specifying constant attributes) while presenting sets of dependencies. Moreover, we treat equivalent sets as one single case

(for two equivalent sets there exists a permutation of attributes transforming one set into another).

2.2. Sets of functional dependencies for small relation schemata. As illustrated in the Introduction, three basic relationship types exist for the binary case: one-to-one, one-to-many, many-to-many. If we fix the role of the two components, the many-to-one version must be included additionally. These relationship types can be described by sets of functional dependencies, treating components as attributes of a relational schema. With the generalization of this concept to higher arities, the different possible types of relationships correspond to the different closed sets of functional dependencies. Reasoning becomes a crucial point since the constraints used for specifying the relationship are not mutually independent. In the following, we present the number of different cases for small arities. Using a PROLOG program based on the simplified formalism and the axiomatization to be discussed in Section 3 we have generated all possible sets up to 5 attributes.

The Ternary Case. The total number of closed sets given three fixed attributes is 45. If permutation of attributes does not matter, sets equivalent up to permutation of attributes are treated as one single case with a representative

Case #	BC → A	AC → B	AB → C	B → A	A → B	C → A	A → C	C → B	B → C	Generating system of functional dependencies	Dimension of attributes $[A]$	$[B]$	$[C]$
#0	0	0	0	0	0	0	0	0	0	$\emptyset$	∞	∞	∞
#1	1	0	0	1	0	0	0	0	0	$\{B \to A\}$	1	∞	∞
#2	1	0	0	1	0	1	0	0	0	$\{B \to A, C \to A\}$	1	∞	∞
#3	1	1	0	0	0	1	0	1	0	$\{C \to B, C \to A\}$	1	1	∞
#4	1	1	0	1	0	1	0	1	0	$\{C \to B, B \to A\}$	1	1	∞
#5	1	1	0	1	1	0	0	0	0	$\{A \to B, B \to A\}$	1	1	∞
#6	1	1	0	1	1	1	0	1	0	$\{C \to B, A \to B, B \to A\}$	1	1	∞
#7	1	1	1	1	1	0	1	0	1	$\{A \to C, A \to B, B \to A\}$	1	1	1
#8	1	1	1	1	1	1	1	1	1	$\{A \to B, B \to C, C \to A\}$	1	1	1
#9	1	1	1	0	0	1	0	1	0	$\{AB \to C, C \to B, C \to A\}$	1	1	2
#10	1	1	0	1	0	0	0	0	0	$\{AC \to B, B \to A\}$	1	2	∞
#11	1	0	0	0	0	0	0	0	0	$\{BC \to A\}$	2	∞	∞
#12	1	1	0	0	0	0	0	0	0	$\{BC \to A, AC \to B\}$	2	2	∞
#13	1	1	1	0	0	0	0	0	0	$\{BC \to A, AC \to B, AB \to C\}$	2	2	2

Table 1. The sets of functional dependencies for the ternary case, grouped by dimensions of attributes

set presented. We get the number of different types of ternary relationships which is 14. Table 1 shows the spreadsheet representation of the sets with their generating systems and attribute dimensions indicated. The graphical presentation of the sets can be found in Section 2.3. These sets were also presented in [9] but we use a different numbering system based on the ordering of attribute dimensions.

Note that the dimension class ($[A] = 1,\ [B] = 2,\ [C] = 2$) is missing since it is a contradictory case. It is easily verified that no valid set of functional dependencies corresponds to this combination of attribute dimensions. If $\mathcal{F}$ were such a set, $[B] = 2$ and $[C] = 2$ would imply $AC \to B,\ AB \to C \in \mathcal{F}$ and no one-dimensional functional dependency would determine B or C. Similarly, $[A] = 1$ would imply that a one-dimensional constraint determining A is in $\mathcal{F}$. Assume $B \to A \in \mathcal{F}$. Since $\{B \to A,\ AB \to C\} \models B \to C$, $B \to C \in \mathcal{F}$ ($\mathcal{F}$ is a closed set). This is a contradiction since $B \to C$ is a one-dimensional constraint determining C and so $[C] = 1$ would hold. A similar contradiction with B arises by assuming $C \to A \in \mathcal{F}$.

The case discussed above is the only contradictory one (up to permutation) for 3 attributes. For 4 attributes, 6 cases arise and for 5 attributes, 28 different contradictory dimension groups exist (up to permutation).

The Quaternary Case. Similarly to the ternary case, sets of functional dependencies for 4 attributes can be presented in the tabular form grouped by value combinations of attribute dimensions. The total number of sets is 2 271; treating equivalent sets as one case we get 165 cases. We show cases of one dimension group only, in Table 2. The complete listing can be found in in [19].

Due to space limitations, the binary representation forms a single column. Grouped bits represent the presence or absence of dependencies in the following order (from left to right): ($BCD \to A$, $ACD \to B$, $ABD \to C$, $ABC \to D$),
($BC \to A$, $AC \to B$, $AB \to C$), ($BD \to A$, $AD \to B$, $AB \to D$),
($CD \to A$, $AD \to C$, $AC \to D$), ($CD \to B$, $BD \to C$, $BC \to D$),
($B \to A$, $A \to B$), ($C \to A$, $A \to C$), ($D \to A$, $A \to D$),
($C \to B$, $B \to C$), ($D \to B$, $B \to D$), ($D \to C$, $C \to D$).

#	Binary representation	Generating system of functional dependencies
	$[A] = 1, [B] = 1, [C] = 2, [D] = 2$	
69	1111 100 010 110 101 00 10 00 00 10 00	$\{AD \to C, BC \to D, C \to A, D \to B\}$
70	1111 101 011 110 101 00 10 00 00 10 00	$\{BC \to D, AB \to C, C \to A, D \to B\}$
71	1111 110 110 011 011 11 00 00 00 00 00	$\{BD \to C, BC \to D, A \to B, B \to A\}$
72	1111 110 110 111 111 11 00 00 00 00 00	$\{CD \to B, BD \to C, BC \to D, A \to B, B \to A\}$

Table 2. Four sets of functional dependencies for the quaternary case

All sets of functional dependencies with five attributes may be represented in a similar form.

Summary of the Number of Closed Sets. Denote by $\mathcal{SD}_n$ the set of closed sets of functional dependencies for a relation schema with n attributes (with constant attributes disallowed). This corresponds to the different relationship types for components with fixed role (asymmetric types counted more than once). Furthermore, let τ be the equivalence relation on these sets classifying them into different types or cases (for two equivalent sets there exists a permutation of attributes transforming one set to another). The number of different classes $(\mathcal{SD}_n/\tau)$ exactly corresponds to the number of relationship types if the attributes do not have a fixed role (an asymmetric relationship type is counted only once). If we allow attributes to be stated as constants (which is, however not likely in schema design), it yields a larger set that is exactly the set of Moore families [21] for n, denoted by $\mathcal{SD}_n^0$ and its equivalence classes $\mathcal{SD}_n^0/\tau$. For each $n \in \mathbb{N}^+$, $\left|\mathcal{SD}_{n+1}^0/\tau\right| = |\mathcal{SD}_{n+1}/\tau| + \left|\mathcal{SD}_n^0/\tau\right|$ easily follows, as well as $\left|\mathcal{SD}_n^0\right| = \sum_{i=0}^{n} \binom{n}{i} |\mathcal{SD}_i|$ where $|\mathcal{SD}_0| = 1$.

With these notations, Table 3 shows the number of different cases for known arities and demonstrates the combinatorial of the search space. The first five rows were computed by our PROLOG program [20, 19] and the third column was also obtained by [22]. The number of Moore families for six elements was presented in [21] and the first column can be calculated from that by the summarization formula above. The number of different equivalence classes for the sixth row is still unknown.

Although the number of different relationship types for n attributes is still unsolved[3], the table shows that the complexity is high even for small arities. This is not surprising from the point of view of constraint sets. However, most work that considers non-binary relationships pays surprisingly little attention to this complexity and some notations used in practice are ambiguous. Therefore, suitable tools are sought for a complete specification of a relationship with functional dependencies.

2.3. The Triangular graphical representation and its generalizations. There have been several proposals (e. g., [3], [25] and [9]) for graphical representation of sets of functional dependencies. Nevertheless, these graphical notations have not made their way into practice and education. The main reason for this failure is the complexity of the representation. We use a notation which

[3]Estimations exist, see [8, 18].

n	$\lvert\mathbb{SD}_n\rvert$	$\lvert\mathbb{SD}_n/\tau\rvert$	$\lvert\mathbb{SD}_n^0\rvert$	$\lvert\mathbb{SD}_n^0/\tau\rvert$
1	1	*1*	2	2
2	4	*3*	7	5
3	45	*14*	61	19
4	2 271	*165*	2 480	184
5	1 373 701	*14 480*	1 385 552	14 664
6	75 965 474 236	*?*	75 973 751 474	?

Table 3. Number of closed sets of functional dependencies for n attributes. The column printed in italics contains the number of basic relationship types of arity n.

reflects the validity of functional dependencies in a simpler and more understandable fashion for small attribute sets [20].

The Triangular Representation for the Ternary Case. A set of functional constraints is represented by a diagram. Functional dependencies and excluded constraints are indicated as nodes of geometrical figures shown on Figure 4. A circle denoting a constraint is always placed at the location of the attribute on the right-hand side of the constraint.

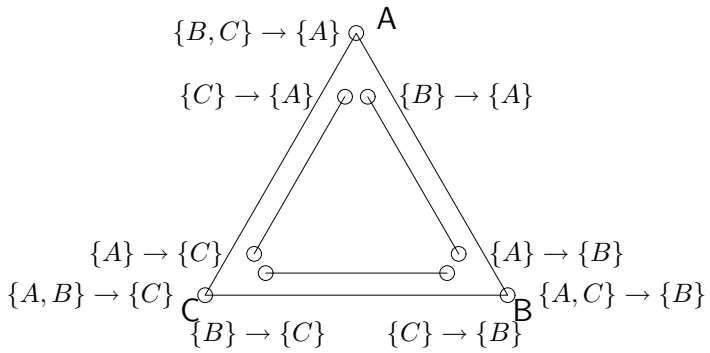

Fig. 4. Triangular representation scheme of sets of functional dependencies for the ternary case

Figure 5 shows three examples of constraint sets represented by graphs. We distinguish two kinds of functional dependencies for $n = 3$:

One-dimensional (singleton left sides): Functional dependencies of the form $A \to B$ are represented by endpoints of binary edges (1D shapes). The filled circle in the left-hand triangle of Figure 5 denotes such a de-

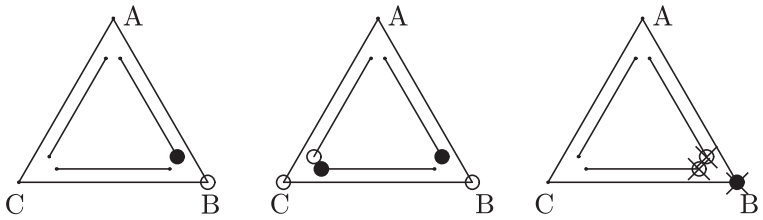

Fig. 5. Examples of the triangular representation

pendency. Constraints such as $A \rightarrow BC$ can be decomposed to canonical functional dependencies $A \rightarrow B$ and $A \rightarrow C$.

Two-dimensional (two-element left sides): Functional dependencies with two-element left-hand sides like $AC \rightarrow B$ cannot be decomposed. They are represented in the triangle (2D shape) on the node relating their right-hand side to the corner, such as the empty circle in the left-hand triangle of Figure 5.

If attributes are allowed to be declared as constants, an extra node is needed for each attribute as a placeholder for a zero-dimensional constraint referring to the attribute (eg. $\emptyset \rightarrow A$). Since we are discussing schema design, this is largely irrelevant.

As shown in Figure 5, *filled circles* represent the basic (initial) dependencies while *empty circles* denote implied dependencies. In the left triangle, the circle of $AC \rightarrow B$ is empty because it is implied by $A \rightarrow B$. The functional dependencies $A \rightarrow B$, $B \rightarrow C$ and their implied functional dependencies are pictured in the middle triangle in a similar way.

We represent also candidates for excluded (negated) functional dependencies by crossed circles in the same fashion for the case when we know that the corresponding functional dependency is not valid in applications. For instance, the negated functional dependency $AC \nrightarrow B$ and the implied negated functional dependencies $A \nrightarrow B$ and $C \nrightarrow B$ are given in the right-hand picture of Figure **??**.

Nodes without a circle or with a small circle correspond to unknown constraints. The specification of a constraint set is complete if all implied constraints are denoted and each node is covered by either a circle or a crossed circle. Each complete set can be actually viewed as a closed set of functional dependencies, using the closed world assumption (everything that is not specified as valid is treated as invalid).

Graphs of all different ternary cases of closed sets (relationship types presented on Table 1) are shown in Figure 6.

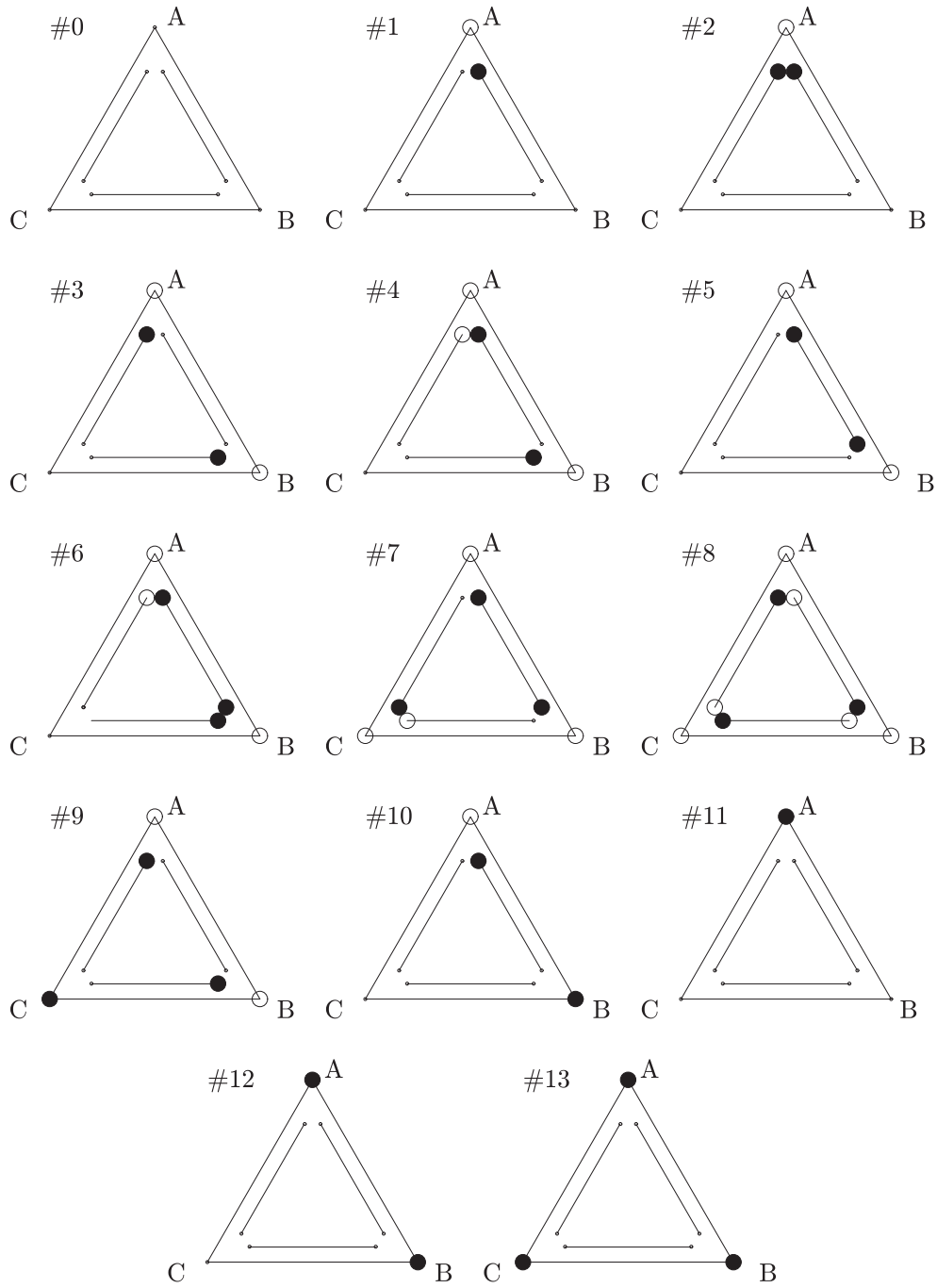

Fig. 6. All sets of functional dependencies in ternary relationship types

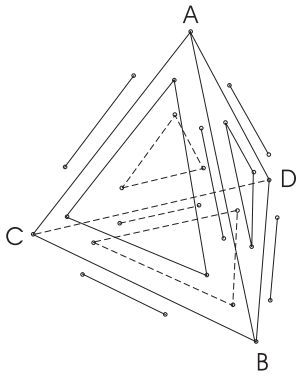

Fig. 7. A tetrahedron as the 3D graphical representation for four attributes (dashed lines indicate invisible edges from the front)

Higher-Arity Generalizations of the Triangular Representation. Considering the triangular representation for the ternary case, it is viewed as a triangle with its three edges repeated (or drawn separately). Each vertex of the triangle as well as each endpoint of the (repeatedly or separately drawn) edges corresponds to a placeholder of a constraint of the matching dimension (1D for edge nodes and 2D for triangular nodes). It is straightforward that the quaternary case contains four nested ternary cases with their one-dimensional parts (edges) shared. Additionally, three-dimensional constraints can be represented as vertices of a three-dimensional shape which is actually a tetrahedron. In this way we get a representation in 3D space, where each node is a placeholder of a functional dependency or an excluded constraint (see Figure 7). For better visibility, separate edges are drawn outside the tetrahedron where possible.

Replacing the 3-dimensional tetrahedron with a square, another version more suitable for 2-dimensional presentation can be constructed with the nested 4 triangles and 6 shared edges drawn inside of it and rearranged.

We have presented the quaternary cases of the dimension class ($[A] = [B] = 1$, $[C] = [D] = 2$) in Table . They are presented in these graphical forms in Figure 8 as an example.

This representation can be generalized to the case of 5 attributes. However, for more attributes, the graph becomes rather complex due to the combinatorical explosion. The more attributes (n) we have, the higher the dimension ($n-1$) of the space is where a simpler and symmetric generalization of triangular representation exists. Complexity can be handled by seeking possible decompositions or by looking at the appropriate translations of schemes ([14], [15], [16]).

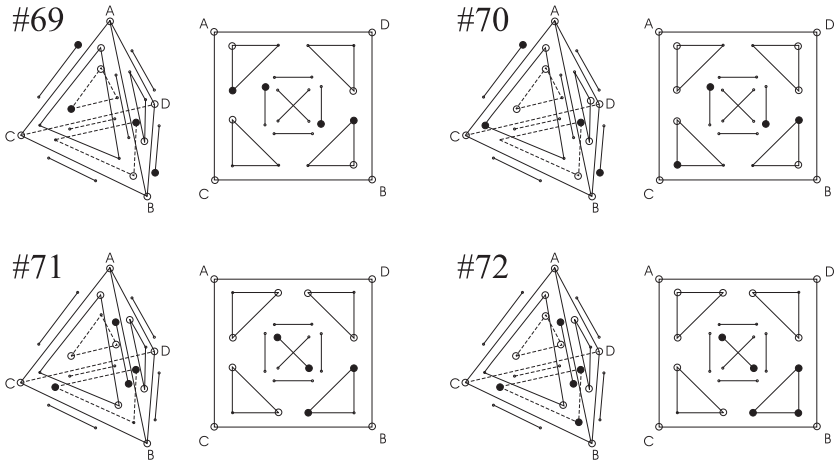

Fig. 8. The tetrahedral and quadratic graphical representations of sets with $[A] = [B] = 1$, $[C] = [D] = 2$ (numbers of sets refer to the spreadsheet representation presented in Section 2.2)

3. Reasoning methods and algorithms.

3.1. Implication systems for functional constraints.

The extended armstrong implication system. Functional dependencies are traditionally axiomatized by the Armstrong implication system [2, 18] with its extended version for negated dependencies [24]:

Axiom

$$XY \to Y$$

Rules

$$(1)\quad \frac{X \longrightarrow Y}{XVW \longrightarrow YV} \qquad (2)\quad \frac{X \longrightarrow Y, Y \longrightarrow Z}{X \longrightarrow Z}$$

$$(3)\quad \frac{X \longrightarrow Y, X \not\to Z}{Y \not\to Z}$$

$$(4)\quad \frac{X \not\to Y}{X \not\to YZ} \qquad (5)\quad \frac{XZ \not\to YZ}{XZ \not\to Y}$$

$$(6)\quad \frac{X \longrightarrow Z, X \not\to YZ}{X \not\to Y} \qquad (7)\quad \frac{Y \longrightarrow Z, X \not\to Z}{X \not\to Y}$$

However, this system has inherent redundancy by considering trivial and non-singleton (right-sided) dependencies[4]. Its main drawback is that in some cases one needs to use the axiom and derive trivial constraints in order to derive a non-trivial target. It adds unnecessary complexity to the constraint implication, which leads to inconvenience for many designers in using the functional dependency notation.

The desired syntactical restriction to nontrivial and singleton dependencies is not compatible with these rules. The restriction can only work if there is an alternative sound and complete axiomatization for the restricted syntax. We present our set of rules that solve this problem in the following subsection.

The ST and PQRST implication systems. In the following rules, Y denotes a set of attributes (allowed to be empty) and A, B, C are different attributes not occurring in Y.

$$(S)\ \frac{Y \to B}{YC \to B} \qquad (T)\ \frac{Y \to A,\ YA \to B}{Y \to B} \qquad (P)\ \frac{YC \nrightarrow B}{Y \nrightarrow B}$$

$$(Q)\ \frac{Y \to A,\ Y \nrightarrow B}{YA \nrightarrow B} \qquad (R)\ \frac{YA \to B,\ Y \nrightarrow B}{Y \nrightarrow A} \qquad (\Box)\ \neg(Y \to B,\ Y \nrightarrow B)$$

- The *ST implication system* for positive constraints contains rules (S) and (T) and no axioms.
- The *PQRST implication system* for both negative and positive constraints has all the presented rules and the symbolic axiom ($\Box$), which is used for indicating contradiction.

These systems are proved as sound and complete for the appropriate universes of dependencies [19][5]. There is no need (nor any possibility) to derive trivial or nonsingleton constraints in order to derive any implication. Moreover, seeking for possible instantiations is more efficient compared to the Armstrong axiomatization because there is always only one attribute set in each rule and at most two different attributes that must not occur in the set.

The U Implication System. The following pseudotransitivity rule schema forms a sound and complete axiomatization for (singleton, non-trivial)

[4] A non-singleton functional dependency can be decomposed into singletons. A non-singleton negated dependency, however, represents a disjunction. We do not consider such dependencies since their relevance is usually not high and by using our implication system they are not needed as intermediate results either (during derivation of any singleton constraint) [19].

[5] For contradictory cases, $\Box$ can be derived.

functional dependencies by itself. It allows a deduction with fewer steps than by using the ST implication system. Consider the following rule. The letters X and Y denote disjoint sets of attributes (allowed to be empty) while A_i's ($i \in [1..k]$, $k \in \mathbb{N}_0$) refer to distinct attributes not occurring in sets X and Y.

$$\text{(U)}\ \frac{XY \to A_1, \cdots, XY \to A_k,\ YA_1 \cdots A_k \to B}{XY \to B}$$

It can also be extended by other rule schemes to cope with negated dependencies as well [19].

3.2. Graphical reasoning on sets of functional dependencies. The rules presented above can directly be applied for deducing the consequences of a set of constraints for small schemata given in terms of the graphical or spreadsheet representation. We will focus on the ST and PQRST systems.

Graphical Rules for the Triangular Representation. The rules of the PQRST implication system support graphical reasoning by their graphical patterns shown in Figure 9 for the triangular representation ($Y = \{C\}$). The small black arrows indicate support (necessary context) while the large grey arrows show the implication effects. Rule (S) is a simple extension rule (extension of the left-hand side) and rule (T) can be called "rotation rule" (rotating a dependency, i. e., changing its right-hand side by the support of a dependency one dimension higher) or "reduction rule" (alternative interpretation: reducing the determinant of a dependency by a lower-dimension support). We prefer the rotation interpretation.

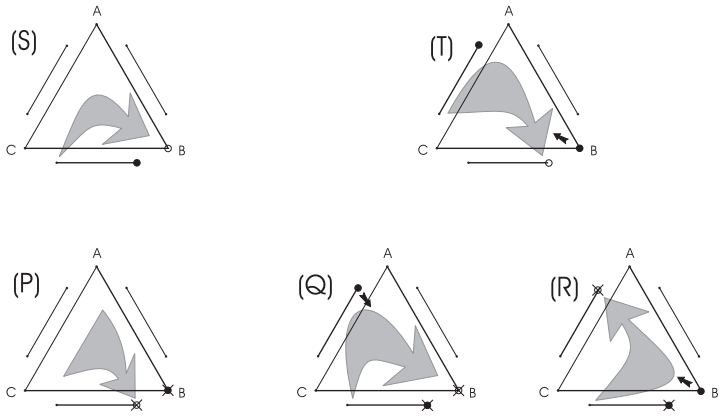

Fig. 9. Graphical versions of rules (S), (T) and (P), (Q), (R)

For excluded functional constraints, rule (Q) acts as the extension rule (needs support of a positive constraint, i. e., functional dependency) and (R) as the rotation rule (needs a positive support too). These two rules can also be viewed as negations of rule (T). Rule (P) is the reduction rule for excluded functional constraints, with the opposite effect in contrast to rule (Q) (but without the need of support). Rule (P) is also viewed as the negation of rule (S).

These graphical rules can be generalized to cases of higher dimensions, where the number of attributes is greater than 3. In such a case, a single rule may have different patterns (e. g., depending on the size of the attribute set Y or the layout of the graph, see [19]).

Recall Figure 5 for three examples of the ternary case. The triangle on the left-hand side shows an example of the application of graphical (triangular) rule (S). On the right-hand side, rule (P) is used twice while in the middle rule (S) is used twice followed by (T). The middle also demonstrates that no explicit rule for transitivity is needed. For an example of the quaternary case, Figure 10 shows how transitivity can be simulated with these rules for the non-singleton case $\{C \to BD,\, BD \to A\} \vdash C \to A$ in both quaternary representations. $C \to BD$ is first decomposed into singleton constraints $\{C \to B,\, C \to D\}$. The numbers

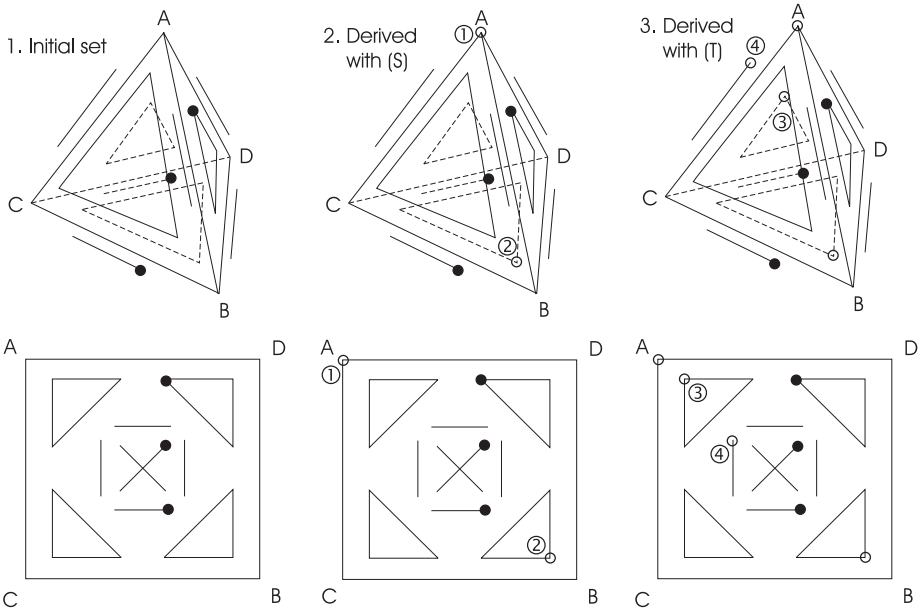

Fig. 10. An example of tetrahedral or quadratic representation and reasoning: Simulating transitivity (numbers show a possible order of deduction)

in the figure show a possible order of deduction: 1. $BD \to A \vdash_{(S)} BCD \to A$; 2. $C \to B \vdash_{(S)} CD \to B$; 3. $CD \to B$ (supported by) $BCD \to A \vdash_{(T)} CD \to A$; 4. $C \to D$ (supported by) $CD \to A \vdash_{(T)} C \to A$. Note that the set is not closed.

When attributes are allowed to be declared as constants, graphical rules for zero-dimensional constraints must be introduced. The implication systems ST and PQRST are capable of handling this type of constraints and the graphical representations can easily be extended (one extra vertex for each attribute) as well as the graphical patterns of derivation rules.

Elicitation of the full knowledge by the ST and STRPQ algorithms. The implication systems introduced above have the advantage of the existence of a specific order of rules which provides a complete algorithmic method for getting all the implied functional dependencies and excluded functional constraints starting with an initial set, allowing one to determine the possible types of relationships the initial set of dependencies defines:

1. Starting with the given initial set of non-trivial, singleton functional dependencies and excluded functional constraints as input,
2. extend the determinants of each dependency using rule (S) as many times as possible, then
3. apply rule (T) until no changes occur,
4. apply rule (R) until no changes occur,
5. reduce and extend the determinants of excluded constraints using rules (P) and (Q) as many times as possible.
6. Output the generated set.

The above method is called *STRPQ algorithm* and can be used for reasoning on sets of functional constraints, especially in terms of the graphical and spreadsheet representations. For positive dependencies only, steps 4 and 5 can be skipped, resulting in the *ST algorithm*. It is proven that this yields a complete method [19].

It can be fine-tuned by taking dimensions into account: start with lower-dimensional instantiations of rule (S) and move towards higher dimensions. When applying rule (T) the opposite should be done: start with the highest-dimensional rotations possible and end with the lowest-dimensional.

BC → A	AC → B	AB → C	B → A	A → B	C → A	A → C	C → B	B → C	Implication impact of detected functional constraints
.	$\vdash 1$	.	.	1	.	.	.	.	(S) $A \to B \vdash AC \to B$
.	.	1	.	1	.	$\vdash 1$	.	.	(T) $AB \to C,\ A \to B \vdash A \to C$
.	0	.	.	$\vdash 0$	.	.	$\vdash 0$	.	(P) $AC \nrightarrow B \vdash A \nrightarrow B,\ C \nrightarrow B$
.	.	$\vdash 0$	.	1	.	0	.	.	(Q) $A \to B,\ A \nrightarrow C \vdash AB \nrightarrow C$
.	.	1	.	$\vdash 0$	.	0	.	.	(R) $AB \to C,\ A \nrightarrow C \vdash A \nrightarrow B$

Table 4. Example of the spreadsheet derivation of functional constraints: Rules of the PQRST implication system in the spreadsheet form

3.3. Extension to spreadsheet reasoning. Let us consider the spreadsheet representation for three attributes. Generalization of the following issues for higher number of attributes is straightforward.

To use the spreadsheet for reasoning, we extend the notation to allow the same distinction for the state of constraints as the small circles and empty/filled circles in the graphical representation. Let 1 and 0 indicate the functional dependencies and excluded functional constraints of the initial set, respectively. We put a '.' in each of the columns corresponding the constraints whose state is not known.

As we get an implied positive constraint (functional dependency) during the deduction process, we replace the corresponding '.' with $\vdash 1$ if the state of the implied constraint was previously unknown. Similarly, an implied negated constraint is indicated by $\vdash 0$.

Table 4 shows how rules of the PQRST implication system can be represented in the spreadsheet form.

The STRPQ algorithm just presented for the graphical representation provides a possible way for derivation of the full knowledge a partial set holds in terms of the spreadsheet representation as well. The spreadsheet can also be used for deriving contradictions. Contradictions occur whenever new constraints are introduced and the implication system allows to derive the opposite. Other implication systems can also be used.

We indicate a contradiction by the symbol $\lightning$. The first case in Table 5 is an obvious one due to the rule system in the extended Armstrong system (reversed transitivity). The second one is due to rule (Q) of the PQRST system.

The elicitation algorithm presented in the introduction of this paper now has a formal basis. We can now derive from a set of given constraints all constraints that are implied and that are contradicted. We thus obtain a number of

BC → 5 A	AC → B	AB → C	B → A	A → B	C → A	A → C	C → B	B → C	Implication impact of detected functional constraints
.	.	.	.	1	.	0	.	$\vdash 0 \not\vdash 1$	$\{A \to B, A \not\twoheadrightarrow C\} \not\vdash \{B \to C\}$
.		$\vdash 0 \not\vdash 1$	.	1	.	0	.	.	$\{A \to B, A \not\twoheadrightarrow C\} \not\vdash \{AB \to C\}$

Table 5. Deriving contradiction by spreadsheet reasoning

constraints whose validity is still open. Using the approach of [1] we can generate sample data and provide them to the designer with the question whether these data support a certain functional dependency or not. Therefore, sets of functional dependencies can be definitively developed and the open problem stated at the beginning is solved based on graphical and elicitation algorithms.

3.4. Other methods for sets of functional constraints.

Closing an attribute set. If the set is not closed and one wants to know the closure of an attribute set X, three methods arise using the graphical (or spreadsheet) representation, each having a strong connection with the well-known closure algorithm. The first step is always dropping dependencies whose right-hand side is in X. Afterwards, attributes are added to X one-by-one. The methods may be combined.

Closing by paths needs the graph to be treated as a hypergraph with dependencies as hyper-edges[6]. To get the closure of an attribute set, one seeks for all attributes that can be reached by a hyper-path starting from X.

Closing by extension means we initially generate dependencies in the form $X \to A$ from each $Y \to A, Y \subset X$ by rule (S) and collect them into a set $\mathcal{F}_X$. Then X is extended by A and $X \to A$ is dropped. In the next step, dependencies of $\mathcal{F}_X$ are extended with A by (S) to match the new X and other dependencies are added from the initial set whose left sides are equal to the new X. This is repeated until $\mathcal{F}_X$ is empty.

Closing by translation is based on the method discussed in [15] adapted to the graphical representation. As an initial step, all basic dependencies are translated by formally removing each attribute of X from the left-hand sides. Some dependencies become zero-dimensional; they can be represented on the graph as extra nodes at the attributes. In this way we get a reduced schema and construct the graphical (or spreadsheet) representation of it. It is especially useful if the schema has many attributes but the initial set X is relatively large

[6] A hyper-edge can have more startpoints (corresponding to the left side of a dependency) but only one endpoint (the right side).

too. We represent only their difference. The constant attributes of the reduced schema – denoted by Y – are in the closure. The next step is a further reduction of the schema by removing attributes of Y and drawing a new graph. The new dependencies are obtained in the same way as before, formally removing elements of Y from the dependencies (this means we simply generate the reductions of them). This process is repeated until no further attributes exist (X is a key) or we get an empty Y (X is not a key).

Determining the validity of a functional dependency and abductive reasoning. For a non-closed set of dependencies, there are basically two options for looking for the validity of a specific dependency by our reasoning framework – just as in any reasoning system.

A straightforward way is to close the attribute set on the left-hand side of the desired dependency using one of the methods just discussed and see whether the attribute on the right-hand side is in the closure.

As an alternative, a target-oriented method can be achieved by declaring the desired constraint as the goal and generating an *and/or structure* of sub-goals the goal can be derived from – e. g., reductions of the functional dependency are alternative sub-goals since if one of them holds, the goal is achieved by the extension rule (S). This method is less efficient, especially if the desired constraint turns out not to be true. However, reversing this gives a possibility of *abductive reasoning*: knowing the goal is true as a basic dependency, the designer can seek for possible causes and decide whether or not to reconsider this basic constraint as implied from some newly found and declared basic dependencies ('explanations') and fine-tune the set of constraints by them.

4. Conclusion and open problems. We have proposed methods for reasoning on sets of functional constraints using a graphical or spreadsheet representation that can help a database designer to achieve complete and unambiguous specification of schemata, especially relationships of higher arity. The main focus is on considering sets of constraints as a whole instead of constraints one-by-one as in the traditional functional dependency notation. We also consider negated (excluded) dependencies to explicitly allow specification of invalidity of functional dependencies. Inherent redundancy of the traditional syntax is eliminated by not considering trivial and non-singleton constraints. A simple and powerful implication system PQRST, convenient for the graphical and spreadsheet representations, is taken as a basis for reasoning. There exists a specific order of rule application for deriving all implied dependencies that can be fine-tuned by the dimension of constraints (the size of their left-hand sides). Other tasks have also

been considered and adaptations of some known algorithms sketched in terms of the graphical representation and this reasoning framework. All this points towards the future implementation of these methods in a software tool.

One open problem is developing convenient graphical representations of attribute separation, grouping methods for cases with more attributes or other types of database constraints. Another, still only partially solved problem is determining the number of different closed sets of functional dependencies for $n \geq 6$ attributes[7].

REFERENCES

[1] Albrecht M., E. Buchholz, A. Düsterhöft, B. Thalheim. An Informal and Efficient Approach for Obtaining Semantic Constraints Using Sample Data and Natural Language Processing. In: Proc. Semantics in Databases, LNCS 1358, Springer, Berlin, 1998, 1–28.

[2] Armstrong W. W. Dependency Structures of Data Base Relationships. In: Information Processing 74, Proceedings of IFIP Congress 74 (Ed. J. L. Rosenfeld), Stockholm, Aug. 5–10, 1974, North-Holland, Amsterdam, 1974, 580–583.

[3] Atzeni P., V. De Antonellis. Relational Database Theory. Addison-Wesley, Redwood City, 1993.

[4] Biskup J. Boyce-Codd Normal Forma and Object Normal Forms. Information Processing Letters, **32**, No 1 (1989), 29–33.

[5] Biskup J. Foundations of Information Systems. Vieweg, Wiesbaden, 1995 (in German).

[6] Biskup J., J. Demetrovics, L. O. Libkin, M. Muchnik. On Relational Database Schemes Having a Unique Minimal Key. *J. of Information Processing*, **27** (1991), 217–225.

[7] Biskup J., T. Polle. Decomposition of Database Classes Under Path Functional Dependencies and Onto Contraints. In: Proc. FoIKS'2000, LNCS 1762, Springer, 2000, 31–49.

[8] Burosch G., J. Demetrovics, G. O. H. Katona, D. J. Kleitman, A. A. Sapozhenko. On the Number of Databases and Closure Operations. TCS, **78**, No 2 (1991), 377–381.

[7] Estimations exist, see [8, 18]

[9] Camps R. From Ternary Relationship to Relational Tables: A Case Against Common Beliefs. ACM SIGMOD Record, **31**, No 2 (2002), 46–49.

[10] Chen & Associates. ER-Designer Reference Manual. Baton Rouge, LA, 1986–1989.

[11] Chen P. P. The Entity-Relationship Model: Toward a Unified View of Data. *ACM TODS*, **1**, No 1 (1976), 9–36.

[12] Chen P. P. (ed.) Proc. 1st Int. ER Conf., ER'79: Entity-Relationship Approach to Systems Analysis and Design, Los Angeles, USA, 1979, North-Holland, Amsterdam, 1980.

[13] Codd. E. F. A Relational Model for Large Shared Data Banks. *CACM*, **13**, No 6 (1970), 197–204.

[14] Demetrovics J., N. X. Huy. Structure of Closure in Relational Databases. In: Proceedings of Conference on intelligent management systems, Bulgarian Academy of Sciences, Varna, 1989, 148–154.

[15] Demetrovics J., N. X. Huy. Translations of Relation Schemes and Representations of Closed Sets. *PU. M. A. Ser. A*, **1**, No 3–4 (1990), 299–315.

[16] Demetrovics J., N. X. Huy. Closed Sets and Translations of Relation Schemes. *Computers Math. Applic*, **21**, No 1 (1991), 13–23.

[17] Demetrovics J., G. Katona. Combinatorial Problems of Database Models. In: Colloquia Mathematica Societatis Janos Bolyai 42, Algebra, Combinatorics and Logic in Computer Science, Győr, Hungary, 1983, 331–352.

[18] Demetrovics J., L. O. Libkin, I. B. Muchnik. Functional Dependencies and the Semilattice of Closed Classes. In: Proc. MFDBS'89, LNCS 364, 1989, 136–147.

[19] Demetrovics J., A. Molnar, B. Thalheim. Graphical and Spreadsheet Reasoning for Sets of Functional Dependencies. Technical Report 0404, Kiel University, Computer Science Institute, `http://www.informatik.uni-kiel.de/reports/2004/0404.html`, 2004.

[20] Demetrovics J., A. Molnar, B. Thalheim. Graphical Reasoning for Sets of Functional Dependencies. In: Proceedings of ER 2004, Lecture Notes in Computer Science **3288**, Springer Verlag, 2004, 166–179.

[21] Habib N., L. Nourine. The Number of Moore Families on n=6. *Discrete Mathematics*, **294**, No 3 (2005), 291–296.

[22] Higuchi A. Note: Lattices of Closure Operators. *Discrete Mathematics*, **179** (1998), 267–272.

[23] Thalheim B. Open Problems in Relational Database Theory. *Bull. EATCS*, **32** (1987), 336–337.

[24] Thalheim B. Entity-Relationship Modeling – Foundations of Database Technology. Springer, Berlin, 2000. See also `http://www.informatik.tu-cottbus.de/~thalheim/HERM.htm`.

[25] Yang C.-C. Relational Databases. Prentice-Hall, Englewood Cliffs, 1986.

János Demetrovics
MTA SZTAKI
Computer and Automation Institute of the Hungarian Academy of Sciences
Kende u. 13-17
H-1111 Budapest, Hungary
e-mail: `demetrovics@sztaki.hu`

András Molnár
Department of Information Systems
Faculty of Informatics
Eötvös Loránd University Budapest
Pázmány Péter stny. 1/C
H-1117 Budapest, Hungary
e-mail: `modras@elte.hu`

Bernhard Thalheim
Computer Science and Applied Mathematics Institute
University Kiel
Olshausenstrasse 40
24098 Kiel, Germany
e-mail: `thalheim@is.informatik.uni-kiel.de`

Received March 9, 2009
Final Accepted April 15, 2009