Annales Univ. Sci. Budapest., Sect. Comp. 28 (2008) 213-233

GRAPHS REPRESENTING SETS OF FUNCTIONAL DEPENDENCIES

J. Demetrovics (Budapest, Hungary)
A. Molnár (Budapest, Hungary)
B. Thalheim (Kiel, Germany)

Dedicated to Imre Kátai on his 70th birthday

Abstract. The most popular graphical database design tool is the Entity-Relationship model (ER). Designing a database schema is based on modeling the real world as entities and relationships among them. ER and other common database modeling tools have restricted capabilities for designing a relationship of higher arity. Although a complete and unambigous specification can be achieved by the traditional functional dependencies for relational schemata, use of the traditional formal notation in practice is rare. We propose a simplified formalism and a graphical framework for designing or surveying the properties of a non-binary relationship among entity classes or attributes, based on functional dependencies. Another representation by the semilattice of closed attribute sets can also be used in parallel. We also make an investigation on the number of closed dependency sets.

1. Introduction

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. Actual data, contained in a *database instance* consists 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 a 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 $R, X \rightarrow Y$ states that only those tables are treated as valid instances of R that contain no pair of rows that 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*$\rightarrow$*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 consists of axioms and rules that support reasoning on constraints.

The notion of functional dependencies was introduced in [1] for the relational database model [13], mainly to provide a way for specification of the properties of valid, acceptable instances of a relational schema. 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. [4, 5, 3, 6]). The Entity-Relationship (ER) model (e.g. [11, 12, 10, 26] 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. They are usually binary, but higher arity relationships (e.g. ternary, quaternary) are allowed and should be used whenever convenient. A database schema from an ER graph can automatically be generated. The model allows specification of cardinalities of entities participating in relationships. We focus on relationships that can be described by sets of functional dependencies.

ER and other common modeling tools have restricted capabilities for designing a relationship of higher arity. Therefore, binarization is often performed even if a higher arity relationship would lead to a more suitable model. In most cases, higher arity relationships can be decomposed (normalized) but to reach a proper decomposition, a precise description of the higher arity relationship must be achieved in advance. Complexity of these kinds of relationships can be high and different types of ternary, quaternary relationships are not

characterized (as opposed to well-known binary cases: one-to-one, one-to-many and many-to-many). The complete and unambigous specification can be achieved by recalling database constraints, relationship construction can be made through relation schema design. To achieve this, the database developer must master semantics acquisition. 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 an 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] and [21] we have proposed novel approaches for graphical and spreadsheet representation of sets of functional dependencies for small relation schemata that supports reasoning. In this paper we present the graphical framework, parallel with the semilattice approach of closed attribute sets discussed in [18]. More details on the graphical and spreadsheet representations can be found in our technical report [19].

The proposal of graphical representation of constraint sets provides a possible solution of the problem of defining a pragmatical approach that allows simple representation of and reasoning on database constraints. The constraint acquisition method below can be refined and adopted into this framework. Solution of the problem is crucial since typical algorithms such as normalization algorithms can only generate a correct result if specification is complete, therefore, the database design process may only be complete of all integrity constraints that cannot be derived by those that have already been specified.

The simplified formalism behind our representation allows to determine the number of different closed sets of functional dependencies for small arities, i.e. the number of possible relationship types.

1.1. Functional constraints, excluded functional constraints and their dimension

We use the traditional functional dependency notation with some restrictions. Besides functional dependencies (FDs), we use *excluded functional constraints* (also called *negated functional dependencies*) as well: $X \not\rightarrow Y$ states that the functional dependency $X \rightarrow Y$ is not valid.

In our notation, a *trivial* 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 has 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 representation deals with non-trivial canonical functional dependencies and non-trivial singleton excluded functional constraints only. We can perform this restriction without loosing relevant deductive power on functional constraints.

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 $\mathcal{F}^+$ 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 just 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.

1.2. Constraint set development

The main task is determining 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 1:

1. *Basic step: Design obvious constraints.*

2. *Recursion step: Repeat until the constraint sets* Σ_0 *and* Σ_1 *do not change:*
   - *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.

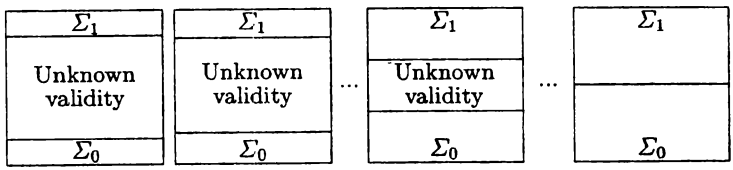

Fig.1. Constraint acquisition process

The number of constraints may however be exponential in the number of attributes [17]. Therefore, specification of the complete set of functional dependencies may be a task that is infeasible. This problem is closely related to another well-known combinatorial problem presented during MFDBS'87 [25] and that is still only partially solved: What is the size of sets of independent functional dependencies for an n-ary relation schema?

2. Sets of functional dependencies and their graphical representations

There have been several proposals (e.g. [2], [28] and [8]) for graphical representation of sets of functional dependencies. Nevertheless, these graphical notations have not made there way into practice and education. The main reason for this failure is the complexity of representation. We use a notation which reflects the validity of functional dependencies in a simpler and better understandable fashion, at least for ternary and quaternary attribute sets [20].

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 to another) when counting the closed sets.

2.1. The ternary case: Triangular representation

A set of canonical, non-trivial functional constraints is represented by a diagram. Functional dependencies and excluded constraints are indicated as nodes of a triangle and endpoints of line sections parallel to its edges. Nodes of the triangle as a 2-dimensional shape are placeholders for 2-dimensional constraints and nodes at the segments correspond to 1-dimensional constraints.[1] Filled circles represent the basic (initial) dependencies while empty circles denote implied dependencies. The circle of a constraint is placed at the location of the attribute on its right-hand side. Crossed circles denote excluded functional constraints in the same fashion. Nodes without a circle or with a small circle correspond to unknown constraints (or excluded constraints if the set is closed and the closed world assumption holds regarding to positive constraints).

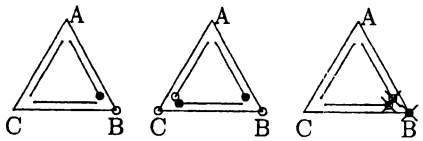

Fig.2. Examples of the triangular representation

Figure 2 shows some examples of the triangular representation. Graphs of all different ternary cases of closed sets (relationship types) can be found in [19]. The functional dependency $\{A\} \rightarrow \{B\}$ and the implied functional dependency $\{A, C\} \rightarrow \{B\}$ are shown in the left part. The functional dependencies $\{A\} \rightarrow \rightarrow \{B\}$, $\{B\} \rightarrow \{C\}$ and their implied functional dependencies are pictured in the middle triangle. The negated functional dependency $\{A, C\} \not\rightarrow \{B\}$ and the implied negated functional dependencies $\{A\} \not\rightarrow \{B\}$ and $\{C\} \not\rightarrow \{B\}$ are given in the right picture.

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 representant set presented. We get the number of different types of ternary relationships which is 14. Graphs of these are shown on Figure 3.

[1] If attributes are allowed to be constants, an extra node is needed for each attribute as a placeholder for zero-dimensional constraint referring to the attribute.

2.2. The quaternary case: Tetrahedral and quadratic 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 correspond to a placeholder of a constraint of matching dimension. It is straightforward that the quaternary case contains four nested ternary cases with their one-dimensional parts (edges) shared. Additionally, three-dimensional constraints ca be represented as vertices of a three-dimensional shape which is actually a tetrahedron. This way we get a representation in 3D space, where each node is a placeholder of a functional dependency or excluded constraint (see Figure 4). For better visibility, separate edges are drawn outside the tetrahedron where possible.

Generalization of the triangular representation can be performed in another direction by constructing planar (2D, *quadratic*) representations. We use the same approach as before in the case of three attributes. An example is displayed in Figure 5 (implication is explained later).

All possible sets of functional dependencies for 4 attributes are presented in a tabular form grouped by value combinations of attribute dimensions in [19]. The total number of sets is 2271, treating equivalent sets as one case we get 165 cases.

2.3. Higher arity cases

The higher-dimensional representation for 5 attributes exists in the 4D space with 5 nested quaternary cases, 10 ternary and 10 binary cases. Each of the edges (corresponding to a binary case) has 3 neighbouring ternary cases (which the binary case is nested in) and 3 neighbouring quaternary cases, while each of the triangles (corresponding to a ternary case) has 2 neighbouring quaternary cases. The frame of the four-dimensional object is shown on Figure 6 as projected to three dimensions. To get the full representation, 5 tetrahedra, 10 triangles and 10 edges formed by the nodes of attributes should be added separately to the figure.

A two-dimensional version can be constructed the same way as the quadratic representation for the quaternary case and is shown on Figure 7. Each node of the graph is a placeholder of a functional constraint (dependency or negated constraint) placeholder is not redundant. Each trapezoid corresponds to a tetrahedron and the bounding pentagon corresponds to the whole body in the 4-dimensional representation. Although this representation may seem complicated (and it actually is, the number of constraint placeholders is 75), one can easily discover which 3 edges belong to a specific triangle or which

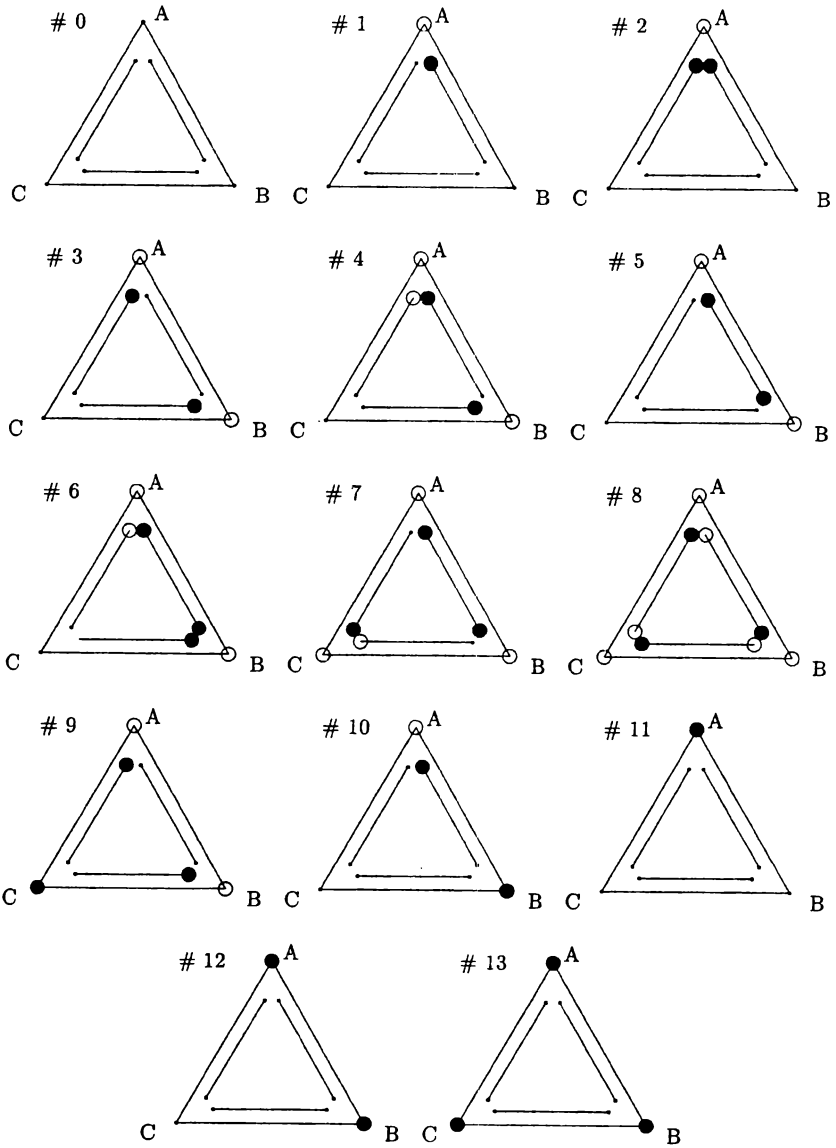

Fig.3. All sets of functional dependencies in ternary relationship types

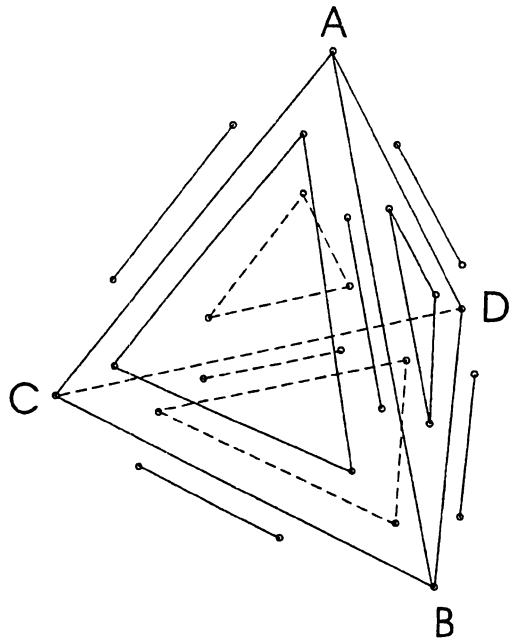

Fig.4. A tetrahedron as the 3D graphical representation for the attributes (stripped lines indicate invisible edges from the front)

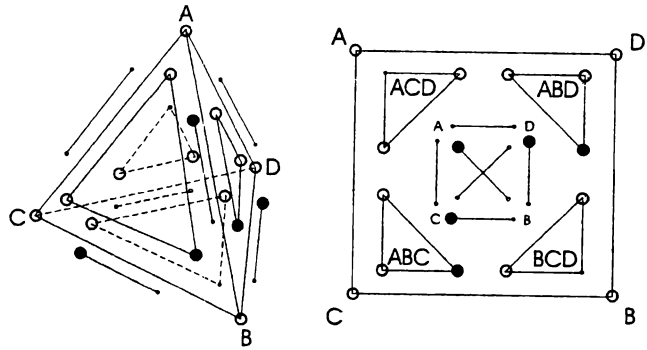

Fig.5. The tetraherdal and quadratic representations of the set generated by $\{B \to C, B \to D, B \to A, AD \to B, AC \to B\}$

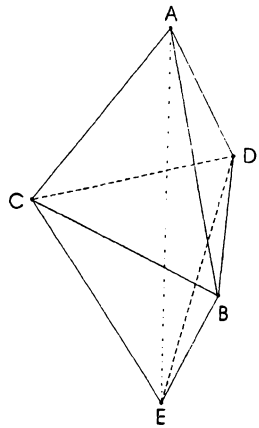

Fig. 6. A three-dimensional projection of the frame of the four-dimensional object for the representation of FD sets over 5 attributes. The five tetrahedra corresponding to nested quaternary cases can easily be discovered, sharing their surface triangles (they represent the ten nested ternary cases) and edges (ten nested binary cases)

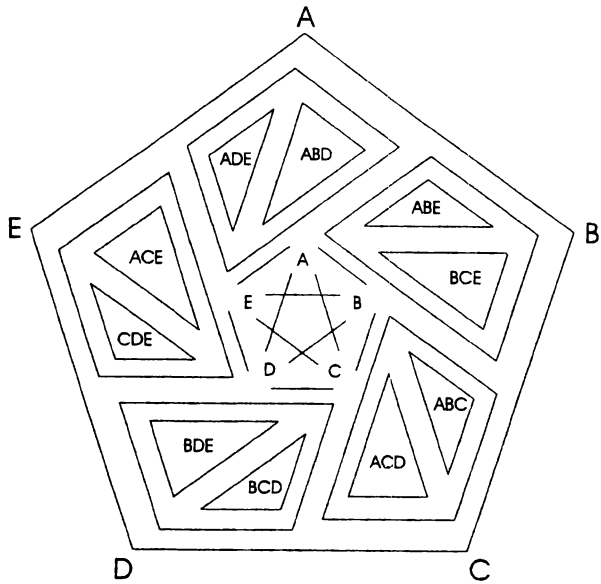

Fig. 7. The pentagonal representation for sets of functional dependencies over 5 attributes

4 triangles belong to a specific trapezoid (or corresponding tetrahedron) by looking at the parallel lines and directions of attributes.

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

2.4. 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 among entity classes with fixed roles (asymmetric types counted more than once). Furthermore, let τ 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 correspond 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 [23] for n, denoted by $\mathcal{SD}_n^0$ and its equivalence classes $\mathcal{SD}_n^0/\tau$. For each $n \in \mathbb{N}^+$, $|\mathcal{SD}_{n+1}^0/\tau| = |\mathcal{SD}_{n+1}/\tau| + |\mathcal{SD}_n^0/\tau|$ easily follows, as well as $|\mathcal{SD}_n^0| = \sum_{i=0}^{n} \binom{n}{i} |\mathcal{SD}_i|$ where $|\mathcal{SD}_0| = 1$.

With these notations, Table 1 shows the number of different cases for known arities and demonstrates the combinatorics of the search space. The first five rows were computed by our PROLOG program [20,19] and the third column was also obtained by [24]. The number of Moore families for six elements was presented in [23] 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,[2] the table shows the complexity is high even for small arities. It is not surprising from the point of view of constraint sets. However, most works that consider non-binary relationships give surprisingly small attention to this complexity and some notations used in practice are ambiguous. Therefore,

[2] Estimations exist, see [7, 18].

suitable tools are sought for a complete specification of a relationship with functional dependencies.

n	$\|\mathcal{SD}_n\|$	$\|\mathcal{SD}_n/\tau\|$	$\|\mathcal{SD}_n^0\|$	$\|\mathcal{SD}_n^0/\tau\|$
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 1. 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

2.5. Representation by the semilattice of closed attribute sets

We present an alternative way of graphical representation of constraint sets that can be used in parallel with the above described triangular notation.

We know from [18] that the set of closed attribute sets wrt a set of functional dependencies is closed under intersection so they form an intersection-semilattice (meet-semilattice, SL). This can be reversed: each set of attribute sets containing the full set (of all the attributes of the schema) that is closed under intersection forms an SL and there exists a set of functional dependencies whose closed attribute sets are exactly the items of the SL.

The semilattice can be represented by a graph [15, 9]: labelled nodes correspond to closed attribute sets and the (nontransitive instances of) set containment are represented by edges. Since the attributes are 'inherited' along the edges, it is enough to indicate the new attributes at each node only. For the sake of clarity, we indicate all the attributes but put the inherited ones in brackets.

Figure 8 shows the semilattice graphs corresponding to examples of Figure 2 with appropriate application of the closed world assumption as follows. In the first two cases where positive constraints are indicated, negated dependencies are assumed for missing constraints. In the third case, negated dependencies

are represented, therefore, each nonnegated candidate constraint is assumed to hold as a functional dependency[3].

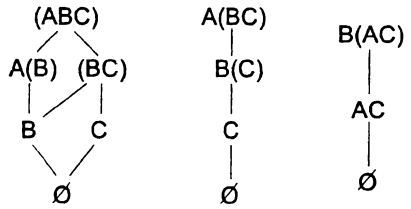

Fig.8. Semilattice graphs of closed attribute sets for examples of Figure 2

The upper bound of the semilattice is the full set of attributes while the lower bound is usually the empty set (the lower bound contains the constant attributes if there are any). Neighbours of the full set are called *coatoms* or *antikeys* ([27], [22], [15]). Equivalent attributes (A, C such that $A \rightarrow C$ and $C \rightarrow A$ both hold) always occur together in a node.

The more dependencies we have, the less vertices the graph has. This is of course not a strict proposition since we may have redundant dependencies. However, adding a nonredundant dependency always destroys at least one closed set, i.e. removes one or more nodes from the graph. The simplest case is a hierarchical chain (if all the attributes contribute to the hierarchy). As the opposite, if all attributes are independent, we get a complete lattice graph. The semilattice graph notation may become complex in some cases while considering the set of functional dependencies is simple. In other cases the situation may be reversed. Therefore, by designing or surveying a relation schema, both representations may be used in parallel, focusing on the one more convenient.

Anyway, there are cases where both ways yield rather complex representations. Dealing with complexity can often be done by performing a suitable decomposition or separation on one of the representations. Separation can either be based on the functional dependency notation (or the graphical representation of it) or on the closed attribute sets approach (semilattice or its graph representation).

[3] If some dependencies remain unknown due to different semantics, a semilattice graph can still be constructed, but some closed attribute sets will be (and must be marked as) uncertain.

3. Reasoning on sets of constraints

3.1. Implication systems for functional constraints

Traditional axiomatization of functional dependencies is the Armstrong implication system and its extended version for negated dependencies [26]. However, these systems have inherent redundancy by considering trivial and non-singleton (right-sided) dependencies[4]. We use a restricted syntax with nontrivial and singleton constraints only and give a sound and complete axiomatization of it.

In the following rules, Y denotes a set of attributes (allowed to be empty) and A, B, C are different attributes not occuring in Y.

$$(S)\ \frac{Y \to B}{YC \to B} \qquad (T)\ \frac{Y \to A,\ YA \to B}{Y \to B} \qquad (P)\ \frac{YC \not\to B}{Y \not\to B}$$

$$(Q)\ \frac{Y \to A,\ Y \not\to B}{YA \not\to B} \qquad (R)\ \frac{YA \to B,\ Y \not\to B}{Y \not\to A} \qquad (U)\ \neg(Y \to B,\ Y \not\to B)$$

- 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 (U), which is used for indicating contradiction.

These systems are proved as sound and complete for the appropriate universes of dependencies [19][5].

3.2. Graphical reasoning on sets of functional dependencies

The rules presented above can directly be applied for deducing consequences of a set of constraints for small schemata given in terms of the graphical representation.

[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 singleton constraints.

[5] For contradictory cases, U can be derived.

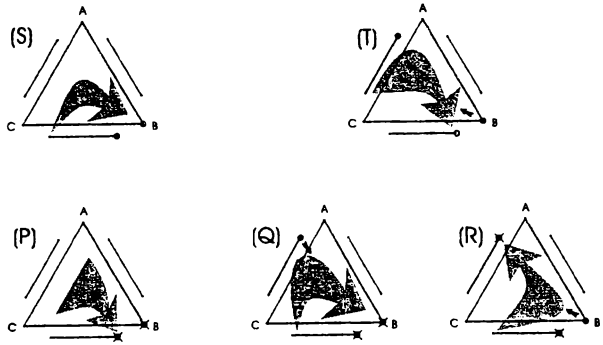

Fig.9. Graphical versions of rules (S), (T) and (P), (Q), (R)

Graphical rules for the triangular representation. Rules of the PQRST implication system support graphical reasoning by their graphical patterns shown on 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 as "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 use the rotation interpretation.

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 than 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 higher dimensional cases, where the number of attributes is more 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 2 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 one

rule (S) is used twice followed by (T). The middle one 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 $\{C \to B,\ C \to D\}$. Numbers on 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.

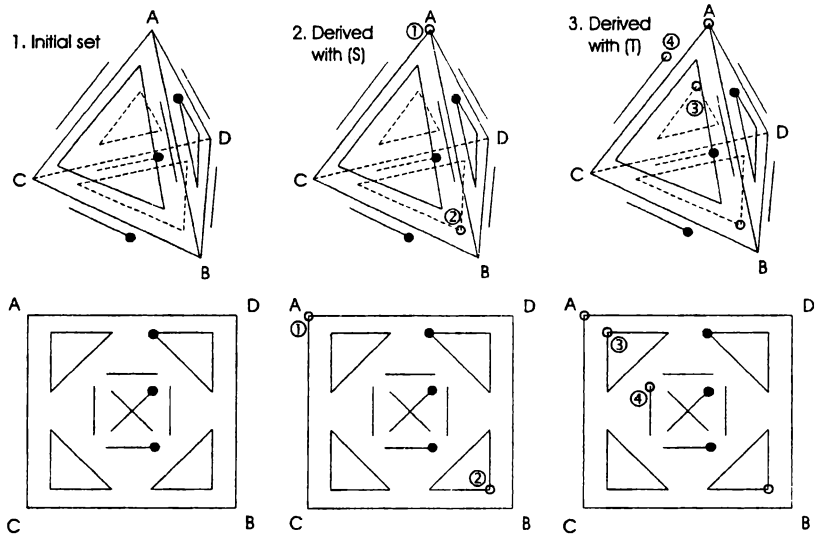

Fig.10. An example of tetrahedral or quadratic representation and reasoning: Simulating transitivity (numbers show a possible order of deduction)

When attributes are allowed to be declared as constants, graphical rules for zero-dimensional constraints are to be introduced. Implication systems ST and PQRST are capable to handle these types 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 representations. For positive dependencies only, steps 4 and 5 can be skipped, resulting the *ST algorithm.*

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.

There are other reasoning methods in terms of the PQRST axiomatization. For example, the method of inserting and deleting a constraint can be found in [21].

3.3. Conversion between representations

Obtaining the closed attribute sets from a closed set of functional dependencies. Whether or not a specific attribute set is closed can be seen from the graphical representation of a closed constraint set. To construct the semilattice graph of all the closed sets needs systematically collecting the closed sets. It is performed on the basis of the following rules:

1. The set of all attributes is always closed.
2. If a set X is closed and no dependency $Y \rightarrow A$ holds, where $Y \subset X$ and $A \in X \setminus Y$, then Y is closed.
3. If two sets X and Y are closed then $X \cap Y$ is closed.

We start with the full set with size n (first rule). Then look at each attribute set with size $n-1$ whether the dependency holds whose left-hand side is that set (simply displayed as the nodes of the outermost shape of the graph, i.e. with the highest dimension). If the dependency does not hold, the set is

added to the closed attribute sets (second rule). Then add the intersections of the obtained sets as closed sets (third rule). This process is repeated with the possible sets in decreasing order by their sizes. The third rule ensures once a closed set is found, determining whether a subset of it is closed needs only to check existence of dependencies completely inside (and not pointing outside) of the set. The whole process can very easily be done also by hand for small relation schemata using the graphical representation.

Refer to the first two cases of Figures 2 and 8 as examples. In the first case, the only two-dimensional dependency holds is $AC \rightarrow B$. So $\{A, B\}$ and $\{B, C\}$ are the closed sets of size 2. Their intersection gives $\{B\}$ as a closed set. What remains is to consider the set $\{A\}$ wrt $\{A, B\}$ and $\{C\}$ wrt $\{B, C\}$. $A \rightarrow B$ holds but $C \rightarrow B$ does not, therefore, $\{C\}$ is a closed set. The empty set is closed since no zero-dimensional dependencies hold. The second case can be transformed easily in a similar way.

Transforming the semilattice graph into a closed set of functional constraints. Given the semilattice of the closed attribute sets, negated functional constraints can be obtained: if a set X is closed, then $X \not\rightarrow A$ holds for each $A \notin X$. These are declared as initial constraints. All other negated constraints can be derived afterwards by the negated reduction rule (P). All the remaining nodes correspond to positive functional dependencies.

Refer to the third case of Figures 2 and 8 as an example. $\{A, C\}$ is the only 'real' closed set (the full set is trivially closed, the closeness of the empty set means no constant attributes must be declared). It causes $AC \not\rightarrow B$ hold, indicated by a crossed circle. By using rule (P), we derive $A \not\rightarrow B$ and $C \not\rightarrow B$. The unmarked nodes should be treated as positive constraints (functional dependencies).

4. Conclusion, future work and open problems

We have proposed a graphical representation and reasoning framework for sets of functional constraints as well as conversion methods between this type of representation and the semilattice of closed attribute sets. The main focus is on considering sets of constraints as a whole instead of constraints one-by-one as in the traditional notation. Inherent redundancy of the traditional syntax is eliminated by not considering trivial and nonsingleton dependencies. 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 to derive all implied dependencies.

Implementation of the discussed methods will give a software tool support for designing relationships by sets of functional constraints. Future work includes incorporating different types of constraints like cardinality constraints, multivalued and inclusion dependencies.

One open problem is developing convenient graphical representations or attribute separation, grouping methods for cases with more attributes. Another, still unsolved, problem is determining the number of different closed sets of functional dependencies for $n \geq 6$ attributes[6]. However, a deeper analysis of the known cases (ternary, quaternary, quinary) is also promising in order to have more sophisticated reasoning facilities on types of relationships.

References

[1] **Armstrong W.W.,** Dependency structures of data base relationships, *Information Processing 74. Proceedings of IFIP Congress 74, Stockholm, Aug. 5-10, 1974,* ed. J.L. Rosenfeld, North-Holland, Amsterdam, 1974, 580-583.

[2] **Atzeni P. and De Antonellis V.,** *Relational database theory,* Addison-Wesley, Redwood City, 1993.

[3] **Biskup J.,** Boyce-Codd normal forma and object normal forms, *Information Processing Letters,* **32** (1) (1989), 29-33.

[4] **Biskup J.,** *Foundations of information systems,* Vieweg, Wiesbaden, 1995. (in German)

[5] **Biskup J., Demetrovics J., Libkin L.O. and Muchnik M.,** On relational database schemes having a unique minimal key, *J. of Information Processing,* **27** (1991), 217-225.

[6] **Biskup J. and Polle T.,** Decomposition of database classes under path functional dependencies and onto constraints, *Proc. FoIKS'2000,* LNCS **1762**, Springer Verlag, 2000, 31-49.

[7] **Burosch G., Demetrovics J., Katona G.O.H., Kleitman D.J. and Sapozhenko A.A.,** On the number of databases and closure operations, *TCS,* **78** (2) (1991), 377-381.

[8] **Camps R.,** From ternary relationship to relational tables: A case against common beliefs, *ACM SIGMOD Record,* **31** (2) (2002), 46-49.

[6] Estimations exist, see [7, 18].

[9] **Caspard N. and Monjardet B.,** The lattices of closure systems, closure operators, and implicational systems on a finite set: a survey, *Discrete Applied Mathematics,* **127** (2003), 241-269.

[10] **Chen & Associates, Baton Rouge, LA,** *ER-designer reference manual,* 1986-1989.

[11] **Chen P.P.,** The entity-relationship model: Toward a unified view of data, *ACM TODS,* **1** (1) (1976), 9-36.

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

[13] **Codd E.F.,** A relational model for large shared data banks, *CACM,* **13** (6) (1970), 197-204.

[14] **Demetrovics J. and Huy N.X.,** Structure of closure in relational databases, *Conference on intelligent management systems, Bulgarian Academy of Sciences, Varna, 1989,* 148-154.

[15] **Demetrovics J. and Huy N.X.,** Translations of relation schemes and representations of closed sets, *PUMA Ser.A,* **1** (3-4) (1990), 299-315.

[16] **Demetrovics J. and Huy N.X.,** Closed sets and translations of relation schemes, *Computers Math. Applic.,* **21** (1) (1991), 13-23.

[17] **Demetrovics J. and Katona G.O.H.,** Combinatorial problems of database models, *Colloquia Mathematica Societatis Janos Bolyai 42, Algebra, Combinatorics and Logic in Computer Science, Győr, Hungary, 1983,* 331-352.

[18] **Demetrovics J., Libkin L.O. and Muchnik I.B.,** Functional dependencies and the semilattice of closed classes, *Proc. MFDB'89,* LNCS **364**, 1989, 136-147.

[19] **Demetrovics J., Molnar A. and Thalheim B.,** *Graphical and spreadsheet reasoning for sets of functional dependencies,* Technical Report 0402, Kiel University, Computer Science Institute, `http://www.informatik.uni-kiel.de/reports/2004/0402.html`, 2004.

[20] **Demetrovics J., Molnar A. and Thalheim B.,** Graphical reasoning for sets of functional dependencies, *Proceedings of ER 2004,* Lecture Notes in Computer Science **3288**, Springer Verlag, 2004, 166-179.

[21] **Demetrovics J., Molnar A. and Thalheim B.,** Relationship design using spreadsheet reasoning for sets of functional dependencies, *Proceedings of ADBIS 2006*, Lecture Notes in Computer Science **4152**, Springer Verlag, 2006, 108-123.

[22] **Demetrovics J. and Thi V.D.,** Some results on functional dependencies, *Acta Cybernetica,* **8** (3) (1988), 273-278.

[23] **Habib N. and Nourine L.,** The number of Moore families on $n = 6$, *Discrete Mathematics,* **294** (3) (2005), 291-296.

[24] **Higuchi A.,** Note: Lattices of closure operators, *Discrete Mathematics,* **179** (1998), 267-272.

[25] **Thalheim B.,** Open problems in relational database theory, *Bull. EATCS,* **32** (1987), 336-337.

[26] **Thalheim B.,** *Entity-relationship modeling - Foundations of database technology,* Springer Verlag, Berlin, 2000. See also `http://www.informatik.tu-cottbus.de/~ thalheim/HERM.htm`.

[27] **Thi V.D.,** Minimal keys and antikeys, *Acta Cybernetica,* **7** (1986), 361-371.

[28] **Yang C.-C.,** *Relational databases,* Prentice-Hall, Englewood Cliffs, 1986.

J. Demetrovics
Computer and Automation Institute
Hungarian Academy of Sciences
Kende u. 13-17.
H-1111 Budapest, Hungary
demetrovics@sztaki.hu

A. Molnár
Department of Information Systems
Eötvös Loránd University
Pázmány Péter sét. 1/C
H-1117 Budapest, Hungary
modras@elte.hu

B. Thalheim
Computer Science and Applied Mathematics Institute
University Kiel
Olshausenstrasse 40
D-24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de