INSTITUT FÜR INFORMATIK

UND PRAKTISCHE MATHEMATIK

Graphical and Spreadsheet Reasoning for Sets of Functional Dependencies

Janos Demetrovics, Andras Molnar, and Bernhard Thalheim

Bericht Nr. 0404
April 2004

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik und Praktische Mathematik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Graphical and Spreadsheet Reasoning for Sets of Functional Dependencies

Janos Demetrovics, Andras Molnar, and Bernhard Thalheim

Bericht Nr. 0404
April 2004

e-mail: demetrovics@sztaki.hu, modras@elte.hu,
thalheim@is.informatik.uni-kiel.de

Dieser Bericht ist als persönliche Mitteilung aufzufassen.[a b]

[a] MTA SZTAKI, Computer and Automation Institute of the Hungarian Academy of Sciences, Kende u. 13-17, H-1111 Budapest, Hungary
[b] Department of Information Systems, Faculty of Informatics, Eotvos Lorand University Budapest, Pazmany Peter stny. 1/C, H-1117 Budapest, Hungary

Abstract

Reasoning on constraint sets is a difficult task. 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. Since the database developer must master semantics acquisition, tools and approaches are still sought that support reasoning on sets of constraints. We propose novel approaches for presentation of sets of functional dependencies based on specific graphs and spreadsheets. These approaches may be used for the elicitation of the full knowledge on validity of functional dependencies in relational schemata.

Chapter 1

Design Problems During Database Semantics Specification and Their Solution

Specification of database structuring is based on three interleaved and dependent parts:

Syntactics: Inductive specification of structures uses a set of base types, a collection of constructors and an theory of construction limiting the application of constructors by rules or by formulas in deontic logics. In most cases, the theory may be dismissed.

Semantics: Specification of admissible databases on the basis of static integrity constraints describes those database states which are considered to be legal.

Pragmatics: Description of context and intension is based either on explicit reference to the enterprize model, to enterprize tasks, to enterprize policy, and environments or on intensional logics used for relating the interpretation and meaning to users depending on time, location, and common sense.

Specification of syntactics is based on the database modeling language. Specification of semantics requires a logical language for specification of classes of constraints. Typical constraints are dependencies such as functional, multivalued, and inclusion dependencies, or domain constraints. Specification of pragmatics is often not explicit. The specification of semantics is often rather difficult due to the complexity. For this reason, it must be supported by a number of solutions supporting acquisition and reasoning on constraints.

Prerequisites of Database Design Approaches

Results obtained during database structuring are evaluated on two main criteria: **completeness** of and **unambiguity** of specification.

Completeness requires that all constraints that must be specified are found. Unambiguity is necessary in order to provide a reasoning system. Both criteria have found their theoretical and pragmatical solution for most of the known classes of constraints. Completeness is, however, restricted by the human ability to survey large constraint sets and to understand all possible interactions among constraints.

Theoretical Approaches to Problem Solution: A number of normalization and restructuring algorithms have been developed for functional dependencies. We do not know yet simple representation systems for surveying constraint sets and for detecting missing constraints beyond functional dependencies.

Pragmatical Approaches to Problem Solution: A step-wise constraint acquisition procedure has been developed in [Kle98, SYG96, Tha00]. 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 approach leads to the following simple elicitation algorithm:

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*.*

This algorithm can be refined in various ways. Elicitation algorithms know so far are all variation of this simple elicitation algorithm.
The constraint acquisition process based on this algorithm is illustrated in Figure 1.

[bhtp]

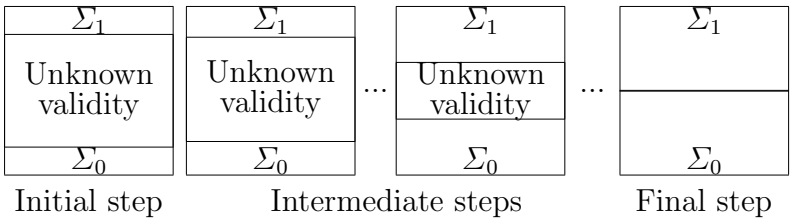

Fig. 1. Constraint Acquisition Process

However, neither the theoretical solutions nor the pragmatical approach provide a solution to problem 1:
Define a pragmatical approach that allows simple representation of and reasoning on database constraints ?

This problem becomes more severe in association with the following problems.

Complexity of Semantics

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 have been specified. Such completeness is not harmful as long as constraint sets are small. The number of constraints may however be exponential in the number of attributes [DK83]. 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 combinatoric problem presented by Janos Demetrovics during MFDBS'87 [Tha87] and that is still only partially solved:
Problem 2. What is the size of sets of independent functional dependencies for an n-ary relation schema?

Inter-Dependence Within a Constraint Set

Constraints such as functional dependencies are not independent from each other. Typical axiomatizations use rules such as the union, transitivity and path rules. Developers do not reason this way. Therefore, the impact of adding, deleting or modifying a constraint within a constraint set is not easy to capture. Therefor, we need a system for reasoning on constraint sets.

Theoretical Approaches to Problem Solution: [Yan86] and [AD93] propose to use a graph-based representation of sets of functional dependencies. This solution provides a simple survey as long as constraints are simple, i.e., use singleton sets for the left sides. [Tha02] proposes to use a schema architecture by developing first elementary schema components and constructing the schema by application of composition operations which use these components. [DLM89] propose to construct a collection of interrelated lattices of functional dependencies. Each lattice represents a component of [Tha02]. The set of functional dependencies is then constructed through folding of the lattices.

Pragmatical Approaches to Problem Solution: [Hal95] proposes to use a fact-based approach instead of modeling of attributes. Elementary facts are 'small' objects that cannot be decomposed without loosing meaning.

We, thus, must solve **problem 3.**
Develop a reasoning system that support easy maintenance and development of constraint sets and highlight logical inter-dependence among constraints.

Instability of Normalization

Normalization is based on the completeness of constraint sets. This is impractical. Constraint sets tend to be incomplete. There are three categories of constraint sets: *'important' constraints* which are always specified, *common sense constraints* which are implicitly assumed but not specified, and *'deep' constraints* which are hard to discover.
The database designer should develop constraints for all three categories. Database design tools can support completeness. However, incompleteness of specification should be considered the *normal* situation. Therefore, normalization approaches should be robust with regard to incompleteness.
Problem 4.[Tha00] Find a normalization theory which is robust for incomplete constraint sets or robust according to a class of changes in constraint sets.

Problems That Currently Defy Solution

Dependency theory consists of work on about 95 classes of dependencies. All these dependencies have been treated in a separate manner. There are very few classes of dependencies that have

been treated together. Moreover, properties of sets of functional dependencies remain still unknown.

In most practical cases several negative results obtained in the dependency theory do not restrict the common utilization of several classes. The reason for this is that the used constraint sets do not have these properties. Therefore, we need other classification principles for describing 'real life' constraint sets.

Problem 5.[Tha00] Classify 'real life' constraint sets which can be easily maintained and specified.

This problem is related to one of the oldest problems in database research expressed by Joachim Biskup in the open problems session [Tha87] of MFDBS'87:

Problem 6. Develop a specification method that supports consideration of sets of functional dependencies and derivation of properties of those sets.

Outline of the Paper and the Kernel Problem Behind Open Problems

The six problems above can be solved on one common basis:

Find a simple and sophisticated representation of sets of constraints that supports reasoning on constraints.

This problem is infeasible in general. Therefore, we provide first a mechanism to reason on sets of functional dependencies defined on small sets of attributes. This mechanism is based on two representations: spreadsheets and geometrical figures such as polygons or tetrahedrons. Next we demonstrate the representation for attribute sets consisting of three, four or five attributes. Finally we introduce the implication system for graphical and spreadsheet representations and show how these representations lead to a very simple and sophisticated treatment of sets of functional dependencies.

Chapter 2

Sets of Functional Dependencies for Small Relation Schemata

2.1 Universes of Functional Constraints

Additionally to functional dependencies, we consider *excluded functional constraints* (*negated functional dependencies*) in the form $X \not\longrightarrow Y$ stating that the functional dependency $X \longrightarrow Y$ is not valid. Certainly, excluded constraints are not needed for the presentation of all possible relationship types (different sets of functional dependencies). However, the two kinds of constraints together can be used for representing partial knowledge on which of the possible functional dependencies hold and can act as the universe (domain) of a formal system providing a reasoning facility. Our graphical and spreadsheet representations allow indication of both types of constraints.

We use the common notation XY for $X \cup Y$ for two sets of attributes X and Y. For a single attribute A, the set $\{A\}$ may be denoted by A when it causes no confusion.

Treating sets of functional constraints becomes simpler if we avoid dealing with obviously redundant constraints. Let us consider two well-known special types of constraints first. A *canonical (singleton)* functional dependency or a singleton excluded functional constraint has exactly one attribute on its right-hand side. 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. The graphical and spreadsheet representations we present in this paper deal with non-trivial canonical functional dependencies and non-trivial singleton excluded functional constraints only. It will be shown that we do not loose relevant deductive power applying this restriction to the universe of functional constraints. We introduce a formal system in Section 4 corresponding to the constraints represented, allowing the derivation of all non-trivial, singleton implications of a set of constraints without the need to include any non-singleton or trivial constraints in the proofs. This provides a reasoning facility in terms of the representations we present.

We introduce the notations $\mathbb{D}$, $\mathbb{D}^+$ and $\mathbb{D}_c^+$ for the universes of functional dependencies, non-trivial functional dependencies and non-trivial canonical (singleton) functional dependencies, respectively, over a fixed underlying domain of attribute symbols. Similarly, $\mathbb{E}$, $\mathbb{E}^+$ and $\mathbb{E}_c^+$ denote the universes of excluded functional constraints, non-trivial excluded constraints and non-trivial singleton excluded functional constraints (negated non-trivial, canonical dependencies) over the same set of attribute symbols, respectively. The traditional universe of functional constraints (including functional dependencies and excluded constraints) is $\mathbb{D} \cup \mathbb{E}$ while our graphical and spreadsheet representations deal with sets of constraints over $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$

which is proper as briefly discussed.

In most of the cases, we focus on *closed* sets of functional dependencies. A finite set $\mathcal{F} \subset \mathbb{D}_c^+$ is closed iff $\mathcal{F}^+ = \mathcal{F}$ where $\mathcal{F}^+$ is the closure of $\mathcal{F}$, ie. $\mathcal{F}^+ = \{\delta \in \mathbb{D}_c^+ \mid \mathcal{F} \vDash \delta\}$.

2.2 The Notion of Dimension

For the classification of functional constraints and the attributes they refer to, we introduce the notion of dimension first. Dimension of a constraint is simply the size of its left-hand side, ie. the number of attributes on its left-hand side. Note that after getting rid of the non-singleton constraints, each constraint we treat has exactly one attribute on its right-hand side. Since we also excluded the trivial constraints from our system, the left-hand side and the right-hand side of a constraint are always disjoint. Therefore, it makes sense for a specific attribute to consider the minimal determinant of it, ie. the left-hand side of a constraint with that attribute on its right-hand side and its left-hand side minimal. The size of this minimal determinant is defined as the dimension of the attribute. We put these definitions to a more precise form.

For a functional dependency $X \to A \in \mathbb{D}_c^+$ denote by $[X \to A]$ its *dimension*, defined as

$$[X \to A] \stackrel{\text{def}}{=} |X|$$

(dimension of an excluded functional constraint can be defined similarly). For a single attribute A, given a set of functional dependencies $\mathcal{F} \subset \mathbb{D}_c^+$, *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 $\mathcal{F}^+$ [1].

Closed sets of functional dependencies (and the corresponding relationship types) can be classified according to the dimensions of the attributes (see Section 2.3). As will be seen later in Section 3, this notion of dimension closely relates to the graphical representation we present.

2.3 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 abscence of the corresponding dependency regarding to 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. Section 6 will show how implication of functional constraints can be performed using the spreadsheet representation.

For a binary relation over the attributes A and B, the possible functional dependencies in $\mathbb{D}_c^+$ are $B \to A$, $A \to B$, $\emptyset \to B$ and $\emptyset \to A$. The first two are one-dimensional, while the rest is zero-dimensional. It is easy to verify that the number of possible (closed) sets is 7 when the roles of A and B are different.

[1] An alternative definition would be $[A] = n$ (where n is the number of attributes considered) if no $X \to A$ exists. However, the notation ∞ emphasizes that A is independent of other attributes and makes the definition independent of n. Note that by excluding the trivial constraints from the system, $\mathcal{F}^+$ contains constraints referring to the attributes occuring in the constraints of $\mathcal{F}$ only, so the closure becomes also independent of the number of actual attributes n (which can be greater than the number of attributes occuring in $\mathcal{F}$).

Since our main focus is on schema design and relationship types, we ignore cases with zero-dimensional constraints (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). This way the number of possible cases for the binary case becomes 3. To give a simple example for the spreadsheet representation, we put them into the form of Table 1 with the conventional notation of functional dependencies as well as the dimensions of attributes indicated.

Case #	FD's $B \to A$	FD's $A \to B$	Conventional notation	Dimension $[A]$	Dimension $[B]$
# 0	0	0	$\emptyset$	∞	∞
# 1	1	0	$\{B \to A\}$	1	∞
# 2	1	1	$\{B \to A,\ A \to B\}$	1	1

Table1. Spreadsheet representation of sets of functional dependencies for binary relations without constant attributes (relationship types, ie. sets equivalent up to attribute permutatuions treated as one case). Dimensions of attributes are also indicated

We will use similar representations for ternary and quadrary relation schemata. We select a representant set for each case of equivalent sets by fixing the order of attributes. Representant sets of cases are ordered and grouped according to the dimensions of attributes. Only those groups (classes) are considered where dimensions of attributes form an increasing sequence ($[A] \leq [B] \leq [C] \leq \cdots$) since we do not get different cases (relationship types) by permuting the attributes.

2.4 Generating and Counting the Number of Sets

Generating and counting the number of (closed) sets for small relation schemata (with 1, 2, 3, 4 and 5 attributes) was performed for both all sets (where a set with its attributes permuted is considered as a different set) and different cases or relationship types (a permutation of attributes does not change the case). Moreover, both calculations were carried out for the cases with and without zero-dimensional constraints as well. A summarizing table can be found in Section 2.8.

Computation was carried out by a back-tracking algorithm using the ST implication system (rules (S) and (T), see section 4.1). Basically, the algorithm works by systematically generating each possible closed set of dependencies. A set is generated by selecting some of the possible one-dimensional dependencies first. A step to one dimension higher is performed by generating dependencies using the extension rule (S) and the remaining possible dependencies of the same dimension give chance for selecting some of them into the set. This step is repeated until the maximal finite dimension is reached ($n - 1$ if the number of attributes is n) or is noticed that the other rule (T) can be applied. In the latter case rule (T) is not applied because it would generate a dependency with one dimension lower but the selection for lower-dimensional dependencies is already fixed. The set is dropped and another selection is made. Since each possible selection is generated, rule (T) is not needed to be applied. The program is written in PROLOG since the logical approach this language provides is an obvious way of formalizing our problem and PROLOG supports backtracking in a natural way.

The refined algorithm works by generating each possible combination for attribute dimensions (increasing sequences), and makes the selections consistent to these dimensions. Treating equivalent sets up to permutation as one single case, we get the number of basically different types (cases). The output of the program is a grouped presentation in terms of the spreadsheet representation.

2.5 The Ternary Case

The total number of closed sets given three fixed attributes is 45. If permutation of attributes does not matter, we get the number of different types of ternary relationships which is 14. Table 2 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 3. These sets were also presented in [**?**] but we use a different numbering system based on the ordering of attribute dimensions.

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

Table2. The sets of functional dependencies for the ternary case, grouped by dimensions of attributes

Note 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 correspond 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 a one-dimensional contstraint 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. Similar contradiction with B arises by assuming $C \to A \in \mathcal{F}$.

2.6 The Quadrary Case

Similarly to the ternary case, sets of functional dependecies for 4 attributes are presented in the tabular form grouped by value combinations of attribute dimensions. Sets equivalent up to permutation of attributes are treated as one single case with a representant set presented (the total number of sets is 2 271, treating equivalent sets as one case we get 165 cases). Due to space limitations, dimension values of attributes are written in separate rows (class headers) and the binary representation forms a single column. Grouped bits represent the presence or absence of dependencies in the following order (from left to right):

$(BCD \to A,\ ACD \to B,\ ABD \to C,\ ABC \to D)$,
$(BC \to A,\ AC \to B,\ AB \to C)$, $(BD \to A,\ AD \to B,\ AB \to D)$,
$(CD \to A,\ AD \to C,\ AC \to D)$, $(CD \to B,\ BD \to C,\ BC \to D)$,
$(B \to A,\ A \to B)$, $(C \to A,\ A \to C)$, $(D \to A,\ A \to D)$,
$(C \to B,\ B \to C)$, $(D \to B,\ B \to D)$, $(D \to C,\ C \to D)$.

Contradictory classes are also indicated in the table.

#	Binary representation	Generating system of functional dependencies
	$[A] = \infty, [B] = \infty, [C] = \infty, [D] = \infty$	
0	0000 000 000 000 000 00 00 00 00 00 00	$\emptyset$
	$[A] = 1, [B] = \infty, [C] = \infty, [D] = \infty$	
1	1000 100 100 000 000 10 00 00 00 00 00	$\{B \to A\}$
2	1000 100 100 100 000 10 00 00 00 00 00	$\{CD \to A, B \to A\}$
3	1000 100 100 100 000 10 10 00 00 00 00	$\{B \to A, C \to A\}$
4	1000 100 100 100 000 10 10 10 00 00 00	$\{B \to A, C \to A, D \to A\}$
	$[A] = 1, [B] = 1, [C] = \infty, [D] = \infty$	
5	1100 100 010 100 100 00 10 00 00 10 00	$\{C \to A, D \to B\}$
6	1100 110 000 100 100 00 10 00 10 00 00	$\{C \to B, C \to A\}$
7	1100 110 100 100 100 00 10 00 10 00 00	$\{BD \to A, C \to B, C \to A\}$
8	1100 110 110 100 100 00 10 00 10 00 00	$\{BD \to A, AD \to B, C \to B, C \to A\}$
9	1100 110 100 100 100 00 10 10 10 00 00	$\{C \to B, C \to A, D \to A\}$
10	1100 110 110 100 100 00 10 10 10 10 00	$\{D \to B, C \to B, C \to A, D \to A\}$
11	1100 110 100 100 100 10 10 00 10 00 00	$\{C \to B, B \to A\}$
12	1100 110 110 100 100 10 10 00 10 00 00	$\{AD \to B, C \to B, B \to A\}$
13	1100 110 100 100 100 10 10 10 10 00 00	$\{C \to B, B \to A, D \to A\}$
14	1100 110 110 100 100 10 10 10 10 10 00	$\{D \to B, C \to B, B \to A\}$
15	1100 110 110 000 000 11 00 00 00 00 00	$\{A \to B, B \to A\}$
16	1100 110 110 100 100 11 00 00 00 00 00	$\{CD \to B, A \to B, B \to A\}$
17	1100 110 110 100 100 11 10 00 10 00 00	$\{C \to B, A \to B, B \to A\}$
18	1100 110 110 100 100 11 10 10 10 10 00	$\{D \to B, C \to B, A \to B, B \to A\}$
	$[A] = 1, [B] = 1, [C] = 1, [D] = \infty$	
19	1110 000 110 110 110 00 00 10 00 10 10	$\{D \to A, D \to B, D \to C\}$
20	1110 100 110 110 110 00 00 10 00 10 10	$\{BC \to A, D \to B, D \to C\}$
21	1110 110 110 110 110 00 00 10 00 10 10	$\{BC \to A, AC \to B, D \to B, D \to C\}$
22	1110 111 110 110 110 00 00 10 00 10 10	$\{BC \to A, AC \to B, AB \to C, D \to B, D \to C\}$
23	1110 100 110 110 110 10 00 10 00 10 10	$\{D \to B, B \to A, D \to C\}$
24	1110 110 110 110 110 10 00 10 00 10 10	$\{AC \to B, D \to B, B \to A, D \to C\}$
25	1110 101 110 110 110 10 00 10 01 10 10	$\{D \to B, B \to C, B \to A\}$

#	Binary representation	Generating system of functional dependencies
26	1110 111 110 110 110 10 00 10 01 10 10	$\{AC \to B, D \to B, B \to C, B \to A\}$
27	1110 100 110 110 110 10 10 10 00 10 10	$\{D \to B, B \to A, C \to A, D \to C\}$
28	1110 110 110 110 110 10 10 10 10 10 10	$\{C \to B, B \to A, D \to C\}$
29	1110 110 110 010 010 11 00 00 00 00 10	$\{A \to B, B \to A, D \to C\}$
30	1110 110 110 110 110 11 00 10 00 10 10	$\{D \to B, A \to B, B \to A, D \to C\}$
31	1110 111 110 010 010 11 01 00 01 00 00	$\{A \to C, A \to B, B \to A\}$
32	1110 111 110 110 110 11 01 00 01 00 00	$\{CD \to B, A \to C, A \to B, B \to A\}$
33	1110 111 110 010 010 11 01 00 01 00 10	$\{A \to C, A \to B, B \to A, D \to C\}$
34	1110 111 110 110 110 11 01 10 01 10 10	$\{D \to B, A \to C, A \to B, B \to A\}$
35	1110 110 110 110 110 11 10 10 10 10 10	$\{C \to B, A \to B, B \to A, D \to C\}$
36	1110 111 110 110 110 11 11 00 11 00 00	$\{A \to C, A \to B, B \to A, C \to A\}$
37	1110 111 110 110 110 11 11 10 11 10 10	$\{A \to C, A \to B, B \to A, C \to A, D \to C\}$
$[A] = 1, [B] = 1, [C] = 1, [D] = 1$		
38	1111 110 110 011 011 11 00 00 00 00 11	$\{A \to B, B \to A, C \to D, D \to C\}$
39	1111 111 111 011 011 11 01 01 01 01 00	$\{B \to A, A \to C, A \to B, A \to D\}$
40	1111 111 111 111 111 11 01 01 01 01 00	$\{CD \to B, B \to A, A \to C, A \to D\}$
41	1111 111 111 011 011 11 01 01 01 01 10	$\{B \to A, A \to B, A \to D, D \to C\}$
42	1111 110 110 111 111 11 10 10 10 10 11	$\{C \to A, A \to B, B \to A, C \to D, D \to C\}$
43	1111 111 111 111 111 11 11 01 11 01 01	$\{C \to B, B \to A, A \to C, A \to D\}$
44	1111 111 111 111 111 11 11 11 11 11 11	$\{C \to B, B \to A, A \to D, D \to C\}$
$[A] = 1, [B] = 1, [C] = 1, [D] = 2$		
45	1111 100 110 110 111 00 00 10 00 10 10	$\{BC \to D, BC \to A, D \to B, D \to C\}$
46	1111 110 110 111 111 00 00 10 00 10 10	$\{BC \to D, BC \to A, AC \to B, D \to B, D \to C\}$
47	1111 111 111 111 111 00 00 10 00 10 10	$\{BC \to D, BC \to A, AC \to B, AB \to C, D \to B, D \to C\}$
48	1111 100 110 110 111 10 00 10 00 10 10	$\{BC \to D, D \to B, B \to A, D \to C\}$
49	1111 110 110 111 111 10 00 10 00 10 10	$\{BC \to D, AC \to B, D \to B, B \to A, D \to C\}$
50	1111 100 110 110 111 10 10 10 00 10 10	$\{BC \to D, D \to B, B \to A, C \to A, D \to C\}$
51	1111 110 110 011 011 11 00 00 00 00 10	$\{BC \to D, A \to B, B \to A, D \to C\}$
52	1111 110 110 111 111 11 00 10 00 10 10	$\{BC \to D, D \to B, A \to B, B \to A, D \to C\}$
$[A] = 1, [B] = 1, [C] = 1, [D] = 3$		
53	1111 000 110 110 110 00 00 10 00 10 10	$\{ABC \to D, D \to A, D \to B, D \to C\}$
$[A] = 1, [B] = 1, [C] = 2, [D] = \infty$		
54	1110 100 010 110 100 00 10 00 00 10 00	$\{AD \to C, C \to A, D \to B\}$
55	1110 101 010 110 100 00 10 00 00 10 00	$\{AB \to C, C \to A, D \to B\}$
56	1110 110 100 100 110 00 10 00 10 00 00	$\{BD \to C, C \to B, C \to A\}$
57	1110 110 110 110 110 00 10 00 10 00 00	$\{BD \to C, AD \to B, C \to B, C \to A\}$
58	1110 110 100 100 110 00 10 10 10 00 00	$\{BD \to C, C \to B, C \to A, D \to A\}$
59	1110 111 000 100 100 00 10 00 10 00 00	$\{AB \to C, C \to B, C \to A\}$
60	1110 111 100 100 110 00 10 00 10 00 00	$\{BD \to C, AB \to C, C \to B, C \to A\}$
61	1110 111 110 110 110 00 10 00 10 00 00	$\{BD \to C, AB \to C, AD \to B, C \to B, C \to A\}$
62	1110 111 100 100 110 00 10 10 10 00 00	$\{AB \to C, C \to B, C \to A, D \to A\}$
63	1110 110 100 100 110 10 10 00 10 00 00	$\{BD \to C, C \to B, B \to A\}$
64	1110 110 110 110 110 10 10 00 10 00 00	$\{BD \to C, AD \to B, C \to B, B \to A\}$
65	1110 110 100 100 110 10 10 10 10 00 00	$\{BD \to C, C \to B, B \to A, D \to A\}$

#	Binary representation	Generating system of functional dependencies
66	1110 110 110 010 010 11 00 00 00 00 00	$\{BD \to C, A \to B, B \to A\}$
67	1110 110 110 110 110 11 00 00 00 00 00	$\{CD \to B, BD \to C, A \to B, B \to A\}$
68	1110 110 110 110 110 11 10 00 10 00 00	$\{BD \to C, C \to B, A \to B, B \to A\}$
$[A] = 1, [B] = 1, [C] = 2, [D] = 2$		
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	$\{CD \to B, BD \to C, BC \to D, A \to B, B \to A\}$
$[A] = 1, [B] = 1, [C] = 2, [D] = 3$		
no valid sets		
$[A] = 1, [B] = 1, [C] = 3, [D] = \infty$		
73	1110 110 000 100 100 00 10 00 10 00 00	$\{ABD \to C, C \to B, C \to A\}$
$[A] = 1, [B] = 1, [C] = 3, [D] = 3$		
no valid sets		
$[A] = 1, [B] = 2, [C] = \infty, [D] = \infty$		
74	1100 100 000 100 100 00 10 00 00 00 00	$\{CD \to B, C \to A\}$
75	1100 100 010 100 100 00 10 00 00 00 00	$\{AD \to B, C \to A\}$
76	1100 100 100 100 100 00 10 00 00 00 00	$\{CD \to B, BD \to A, C \to A\}$
77	1100 100 110 100 100 00 10 00 00 00 00	$\{BD \to A, AD \to B, C \to A\}$
78	1100 100 100 100 100 00 10 10 00 00 00	$\{CD \to B, C \to A, D \to A\}$
79	1100 100 100 100 100 10 00 00 00 00 00	$\{CD \to B, B \to A\}$
80	1100 110 100 000 000 10 00 00 00 00 00	$\{AC \to B, B \to A\}$
81	1100 110 100 100 100 10 00 00 00 00 00	$\{CD \to B, AC \to B, B \to A\}$
82	1100 110 110 000 000 10 00 00 00 00 00	$\{AC \to B, AD \to B, B \to A\}$
83	1100 110 110 100 100 10 00 00 00 00 00	$\{CD \to B, AC \to B, AD \to B, B \to A\}$
84	1100 100 100 100 100 10 10 00 00 00 00	$\{CD \to B, B \to A, C \to A\}$
85	1100 100 110 100 100 10 10 00 00 00 00	$\{AD \to B, B \to A, C \to A\}$
86	1100 100 100 100 100 10 10 10 00 00 00	$\{CD \to B, B \to A, C \to A, D \to A\}$
$[A] = 1, [B] = 2, [C] = 2, [D] = \infty$		
87	1110 000 100 100 110 00 00 10 00 00 00	$\{CD \to B, BD \to C, D \to A\}$
88	1110 010 100 100 110 00 00 10 00 00 00	$\{BD \to C, AC \to B, D \to A\}$
89	1110 011 100 100 110 00 00 10 00 00 00	$\{AC \to B, AB \to C, D \to A\}$
90	1110 100 100 100 110 00 00 10 00 00 00	$\{CD \to B, BD \to C, BC \to A, D \to A\}$
91	1110 110 100 100 110 00 00 10 00 00 00	$\{BD \to C, BC \to A, AC \to B, D \to A\}$
92	1110 111 100 100 110 00 00 10 00 00 00	$\{BC \to A, AC \to B, AB \to C, D \to A\}$
93	1110 100 100 100 110 10 00 00 00 00 00	$\{CD \to B, BD \to C, B \to A\}$
94	1110 100 110 010 010 10 00 00 00 00 00	$\{BD \to C, AD \to B, B \to A\}$
95	1110 100 110 110 110 10 00 00 00 00 00	$\{CD \to B, BD \to C, AD \to B, B \to A\}$
96	1110 100 100 100 110 10 00 10 00 00 00	$\{CD \to B, BD \to C, B \to A, D \to A\}$
97	1110 110 100 000 010 10 00 00 00 00 00	$\{BD \to C, AC \to B, B \to A\}$
98	1110 110 100 100 110 10 00 00 00 00 00	$\{CD \to B, BD \to C, AC \to B, B \to A\}$
99	1110 110 110 010 010 10 00 00 00 00 00	$\{BD \to C, AC \to B, AD \to B, B \to A\}$
100	1110 110 110 110 110 10 00 00 00 00 00	$\{CD \to B, BD \to C, AC \to B, AD \to B, B \to A\}$
101	1110 110 100 100 110 10 00 10 00 00 00	$\{BD \to C, AC \to B, B \to A, D \to A\}$

#	Binary representation	Generating system of functional dependencies
102	1110 100 100 100 110 10 10 00 00 00 00	$\{CD \to B, BD \to C, B \to A, C \to A\}$
103	1110 100 110 110 110 10 10 00 00 00 00	$\{BD \to C, AD \to B, B \to A, C \to A\}$
104	1110 100 100 100 110 10 10 10 00 00 00	$\{CD \to B, BD \to C, B \to A, C \to A, D \to A\}$
$[A] = 1, [B] = 2, [C] = 2, [D] = 2$		
105	1111 100 100 100 111 10 00 00 00 00 00	$\{CD \to B, BD \to C, BC \to D, B \to A\}$
106	1111 110 100 001 011 10 00 00 00 00 00	$\{BD \to C, BC \to D, AC \to B, B \to A\}$
107	1111 110 100 101 111 10 00 00 00 00 00	$\{CD \to B, BD \to C, BC \to D, AC \to B, B \to A\}$
108	1111 110 110 011 011 10 00 00 00 00 00	$\{BD \to C, BC \to D, AC \to B, AD \to B, B \to A\}$
109	1111 110 110 111 111 10 00 00 00 00 00	$\{CD \to B, BD \to C, BC \to D, AC \to B, AD \to B, B \to A\}$
110	1111 100 100 100 111 10 10 00 00 00 00	$\{CD \to B, BD \to C, BC \to D, B \to A, C \to A\}$
111	1111 100 110 110 111 10 10 00 00 00 00	$\{BD \to C, BC \to D, AD \to B, B \to A, C \to A\}$
112	1111 100 100 100 111 10 10 10 00 00 00	$\{CD \to B, BD \to C, BC \to D, B \to A, C \to A, D \to A\}$
$[A] = 1, [B] = 2, [C] = 2, [D] = 3$		
113	1111 000 100 100 110 00 00 10 00 00 00	$\{ABC \to D, CD \to B, BD \to C, D \to A\}$
$[A] = 1, [B] = 2, [C] = 3, [D] = \infty$		
114	1110 100 000 100 100 00 10 00 00 00 00	$\{ABD \to C, CD \to B, C \to A\}$
$[A] = 1, [B] = 2, [C] = 3, [D] = 3$		
no valid sets		
$[A] = 1, [B] = 3, [C] = \infty, [D] = \infty$		
115	1100 100 100 000 000 10 00 00 00 00 00	$\{ACD \to B, B \to A\}$
$[A] = 1, [B] = 3, [C] = 3, [D] = \infty$		
no valid sets		
$[A] = 1, [B] = 3, [C] = 3, [D] = 3$		
no valid sets		
$[A] = 2, [B] = \infty, [C] = \infty, [D] = \infty$		
116	1000 100 000 000 000 00 00 00 00 00 00	$\{BC \to A\}$
117	1000 100 100 000 000 00 00 00 00 00 00	$\{BC \to A, BD \to A\}$
118	1000 100 100 100 000 00 00 00 00 00 00	$\{BC \to A, BD \to A, CD \to A\}$
$[A] = 2, [B] = 2, [C] = \infty, [D] = \infty$		
119	1100 000 000 100 100 00 00 00 00 00 00	$\{CD \to B, CD \to A\}$
120	1100 100 000 100 100 00 00 00 00 00 00	$\{CD \to B, BC \to A\}$
121	1100 100 010 000 000 00 00 00 00 00 00	$\{BC \to A, AD \to B\}$
122	1100 100 010 100 100 00 00 00 00 00 00	$\{CD \to B, BC \to A, AD \to B\}$
123	1100 100 100 100 100 00 00 00 00 00 00	$\{CD \to B, BC \to A, BD \to A\}$
124	1100 110 000 000 000 00 00 00 00 00 00	$\{BC \to A, AC \to B\}$
125	1100 110 000 100 100 00 00 00 00 00 00	$\{CD \to B, BC \to A, AC \to B\}$
126	1100 110 100 000 000 00 00 00 00 00 00	$\{BC \to A, AC \to B, BD \to A\}$
127	1100 110 100 100 100 00 00 00 00 00 00	$\{CD \to B, BC \to A, AC \to B, BD \to A\}$
128	1100 110 110 000 000 00 00 00 00 00 00	$\{BC \to A, AC \to B, BD \to A, AD \to B\}$
129	1100 110 110 100 100 00 00 00 00 00 00	$\{CD \to B, BC \to A, AC \to B, BD \to A, AD \to B\}$
$[A] = 2, [B] = 2, [C] = 2, [D] = \infty$		
130	1110 000 110 010 010 00 00 00 00 00 00	$\{BD \to C, BD \to A, AD \to B\}$

#	Binary representation	Generating system of functional dependencies
131	1110 000 110 110 110 00 00 00 00 00 00	$\{CD \to B, BD \to C, BD \to A, AD \to B\}$
132	1110 100 010 010 000 00 00 00 00 00 00	$\{AD \to C, BC \to A, AD \to B\}$
133	1110 100 100 100 110 00 00 00 00 00 00	$\{CD \to B, BD \to C, BC \to A\}$
134	1110 100 110 010 010 00 00 00 00 00 00	$\{BD \to C, BC \to A, AD \to B\}$
135	1110 100 110 110 110 00 00 00 00 00 00	$\{CD \to B, BD \to C, BC \to A, AD \to B\}$
136	1110 110 100 000 010 00 00 00 00 00 00	$\{BD \to C, BC \to A, AC \to B\}$
137	1110 110 100 100 110 00 00 00 00 00 00	$\{CD \to B, BD \to C, BC \to A, AC \to B\}$
138	1110 110 110 010 010 00 00 00 00 00 00	$\{BD \to C, BC \to A, AC \to B, AD \to B\}$
139	1110 110 110 110 110 00 00 00 00 00 00	$\{CD \to B, BD \to C, BC \to A, AC \to B, AD \to B\}$
140	1110 111 000 000 000 00 00 00 00 00 00	$\{BC \to A, AC \to B, AB \to C\}$
141	1110 111 100 000 010 00 00 00 00 00 00	$\{BD \to C, BC \to A, AC \to B, AB \to C\}$
142	1110 111 110 010 010 00 00 00 00 00 00	$\{BD \to C, BC \to A, AC \to B, AB \to C, AD \to B\}$
143	1110 111 110 110 110 00 00 00 00 00 00	$\{CD \to B, BD \to C, BC \to A, AC \to B, AB \to C, AD \to B\}$
	$[A] = 2, [B] = 2, [C] = 2, [D] = 2$	
144	1111 100 010 010 001 00 00 00 00 00 00	$\{AD \to C, BC \to D, BC \to A, AD \to B\}$
145	1111 110 100 001 011 00 00 00 00 00 00	$\{BD \to C, BC \to D, BC \to A, AC \to B\}$
146	1111 110 110 011 011 00 00 00 00 00 00	$\{BD \to C, BC \to D, BC \to A, AC \to B, AD \to B\}$
147	1111 111 001 001 001 00 00 00 00 00 00	$\{BC \to D, BC \to A, AC \to B, AB \to C\}$
148	1111 111 101 001 011 00 00 00 00 00 00	$\{BD \to C, BC \to D, BC \to A, AC \to B, AB \to C\}$
149	1111 111 111 011 011 00 00 00 00 00 00	$\{BD \to C, BC \to D, BC \to A, AC \to B, AB \to C, AD \to B\}$
150	1111 111 111 111 111 00 00 00 00 00 00	$\{CD \to B, BD \to C, BC \to D, BC \to A, AC \to B, AB \to C, AD \to B\}$
	$[A] = 2, [B] = 2, [C] = 2, [D] = 3$	
151	1111 000 110 010 010 00 00 00 00 00 00	$\{ABC \to D, BD \to C, BD \to A, AD \to B\}$
152	1111 000 110 110 110 00 00 00 00 00 00	$\{ABC \to D, CD \to B, BD \to C, BD \to A, AD \to B\}$
	$[A] = 2, [B] = 2, [C] = 3, [D] = \infty$	
153	1110 000 000 100 100 00 00 00 00 00 00	$\{ABD \to C, CD \to B, CD \to A\}$
154	1110 100 000 100 100 00 00 00 00 00 00	$\{ABD \to C, CD \to B, BC \to A\}$
155	1110 110 000 000 000 00 00 00 00 00 00	$\{ABD \to C, BC \to A, AC \to B\}$
156	1110 110 000 100 100 00 00 00 00 00 00	$\{ABD \to C, CD \to B, BC \to A, AC \to B\}$
	$[A] = 2, [B] = 2, [C] = 3, [D] = 3$	
157	1111 000 000 100 100 00 00 00 00 00 00	$\{ABD \to C, ABC \to D, CD \to B, CD \to A\}$
	$[A] = 2, [B] = 3, [C] = \infty, [D] = \infty$	
158	1100 100 000 000 000 00 00 00 00 00 00	$\{ACD \to B, BC \to A\}$
159	1100 100 100 000 000 00 00 00 00 00 00	$\{ACD \to B, BC \to A, BD \to A\}$
	$[A] = 2, [B] = 3, [C] = 3, [D] = \infty$	
160	1110 100 000 000 000 00 00 00 00 00 00	$\{ACD \to B, ABD \to C, BC \to A\}$
	$[A] = 2, [B] = 3, [C] = 3, [D] = 3$	
	no valid sets	
	$[A] = 3, [B] = \infty, [C] = \infty, [D] = \infty$	
161	1000 000 000 000 000 00 00 00 00 00 00	$\{BCD \to A\}$

#	Binary representation	Generating system of functional dependencies
$[A]=3,[B]=3,[C]=\infty,[D]=\infty$		
162	1100 000 000 000 000 00 00 00 00 00 00	$\{BCD\to A, ACD\to B\}$
$[A]=3,[B]=3,[C]=3,[D]=\infty$		
163	1110 000 000 000 000 00 00 00 00 00 00	$\{BCD\to A, ACD\to B, ABD\to C\}$
$[A]=3,[B]=3,[C]=3,[D]=3$		
164	1111 000 000 000 000 00 00 00 00 00 00	$\{BCD\to A, ACD\to B, ABD\to C, ABC\to D\}$

Table3.: The sets of functional dependencies for the quadrary case

2.7 The Quintary Case

AS the number of attributes is raised to five, the total number of sets increases to more than a million (exactly 1 373 701) while the different types remain relatively low, 14 480. However, this number is still rather high compared to the ternary or quadrary case. We omit the presentation of all the sets themselves, intstead, we give a list of the number of cases for the different classes of attribute dimensions on Table 4. Table 5 shows the cases of class ($[A]=1,[B]=1,[C]=1,[D]=2,[E]=3$) as an example. Grouped bits of the binary (tabular) representation represent the presence or absence of dependencies in the following order (from left to right):
$(BCDE\to A,\ ACDE\to B,\ ABDE\to C,\ ABCE\to D,\ ABCD\to E)$,
$(BCD\to A,\ ACD\to B,\ ABD\to C,\ ABC\to D)$,
$(BCE\to A,\ ACE\to B,\ ABE\to C,\ ABC\to E)$,
$(BDE\to A,\ ADE\to B,\ ABE\to D,\ ABD\to E)$,
$(CDE\to A,\ ADE\to C,\ ACE\to D,\ ACD\to E)$,
$(CDE\to B,\ BDE\to C,\ BCE\to D,\ BCD\to E)$,
$(BC\to A,\ AC\to B,\ AB\to C)$, $(BD\to A,\ AD\to B,\ AB\to D)$,
$(BE\to A,\ AE\to B,\ AB\to E)$, $(CD\to A,\ AD\to C,\ AC\to D)$,
$(CE\to A,\ AE\to C,\ AC\to E)$, $(DE\to A,\ AE\to D,\ AD\to E)$,
$(CD\to B,\ BD\to C,\ BC\to D)$, $(CE\to B,\ BE\to C,\ BC\to E)$,
$(DE\to B,\ BE\to D,\ BD\to E)$, $(DE\to C,\ CE\to D,\ CD\to E)$,
$(B\to A,\ A\to B)$, $(C\to A,\ A\to C)$, $(D\to A,\ A\to D)$, $(E\to A,\ A\to E)$,
$(C\to B,\ B\to C)$, $(D\to B,\ B\to D)$, $(E\to B,\ B\to E)$,
$(D\to C,\ C\to D)$, $(E\to C,\ C\to E)$, $(E\to D,\ D\to E)$.

Class [A]	[B]	[C]	[D]	[E]	Number of cases
∞	∞	∞	∞	∞	1
1	∞	∞	∞	∞	9
1	1	∞	∞	∞	69
1	1	1	∞	∞	189
1	1	1	1	∞	207
1	1	1	1	1	32
1	1	1	1	2	131
1	1	1	1	3	25
1	1	1	1	4	1

Class					Number
[A]	[B]	[C]	[D]	[E]	of cases
1	1	1	2	∞	322
1	1	1	2	2	82
1	1	1	2	3	6
1	1	1	2	4	↯ 0
1	1	1	3	∞	45
1	1	1	3	3	↯ 0
1	1	1	3	4	↯ 0
1	1	1	4	∞	1
1	1	1	4	4	↯ 0
1	1	2	∞	∞	431
1	1	2	2	∞	762
1	1	2	2	2	327
1	1	2	2	3	80
1	1	2	2	4	1
1	1	2	3	∞	103
1	1	2	3	3	4
1	1	2	3	4	↯ 0
1	1	2	4	∞	1
1	1	2	4	4	↯ 0
1	1	3	∞	∞	47
1	1	3	3	∞	5
1	1	3	3	3	2
1	1	3	3	4	↯ 0
1	1	3	4	∞	↯ 0
1	1	3	4	4	↯ 0
1	1	4	∞	∞	1
1	1	4	4	∞	↯ 0
1	1	4	4	4	↯ 0
1	2	∞	∞	∞	144
1	2	2	∞	∞	866
1	2	2	2	∞	1355
1	2	2	2	2	435
1	2	2	2	3	241
1	2	2	2	4	2
1	2	2	3	∞	533
1	2	2	3	3	71
1	2	2	3	4	1
1	2	2	4	∞	4
1	2	2	4	4	↯ 0
1	2	3	∞	∞	233
1	2	3	3	∞	105
1	2	3	3	3	12
1	2	3	3	4	↯ 0
1	2	3	4	∞	1

Class					Number
[A]	[B]	[C]	[D]	[E]	of cases
1	2	3	4	4	↯ 0
1	2	4	∞	∞	2
1	2	4	4	∞	↯ 0
1	2	4	4	4	↯ 0
1	3	∞	∞	∞	32
1	3	3	∞	∞	49
1	3	3	3	∞	46
1	3	3	3	3	18
1	3	3	3	4	1
1	3	3	4	∞	1
1	3	3	4	4	↯ 0
1	3	4	∞	∞	1
1	3	4	4	∞	↯ 0
1	3	4	4	4	↯ 0
1	4	∞	∞	∞	1
1	4	4	∞	∞	↯ 0
1	4	4	4	∞	↯ 0
1	4	4	4	4	↯ 0
2	∞	∞	∞	∞	14
2	2	∞	∞	∞	207
2	2	2	∞	∞	1070
2	2	2	2	∞	1411
2	2	2	2	2	451
2	2	2	2	3	419
2	2	2	2	4	7
2	2	2	3	∞	1171
2	2	2	3	3	287
2	2	2	3	4	8
2	2	2	4	∞	19
2	2	2	4	4	1
2	2	3	∞	∞	646
2	2	3	3	∞	557
2	2	3	3	3	112
2	2	3	3	4	4
2	2	3	4	∞	15
2	2	3	4	4	↯ 0
2	2	4	∞	∞	14
2	2	4	4	∞	1
2	2	4	4	4	↯ 0
2	3	∞	∞	∞	91
2	3	3	∞	∞	278
2	3	3	3	∞	288
2	3	3	3	3	96
2	3	3	3	4	8

Class [A]	[B]	[C]	[D]	[E]	Number of cases
2	3	3	4	∞	18
2	3	3	4	4	1
2	3	4	∞	∞	13
2	3	4	4	∞	1
2	3	4	4	4	↯ 0
2	4	∞	∞	∞	4
2	4	4	∞	∞	1
2	4	4	4	∞	↯ 0
2	4	4	4	4	↯ 0
3	∞	∞	∞	∞	4
3	3	∞	∞	∞	22
3	3	3	∞	∞	60
3	3	3	3	∞	70
3	3	3	3	3	23
3	3	3	3	4	7
3	3	3	4	∞	14
3	3	3	4	4	2
3	3	4	∞	∞	11
3	3	4	4	∞	4
3	3	4	4	4	1
3	4	∞	∞	∞	3
3	4	4	∞	∞	2
3	4	4	4	∞	1
3	4	4	4	4	↯ 0
4	∞	∞	∞	∞	1
4	4	∞	∞	∞	1
4	4	4	∞	∞	1
4	4	4	4	∞	1
4	4	4	4	4	1

Table4.: The number of different sets of functional dependencies for the quintary case, grouped by classes of attribute dimensions. Contradictory classes are indicated by the symbol ↯

#	Binary representation
	...
	$[A] = 1, [B] = 1, [C] = 1, [D] = 2, [E] = 3$
1068	11111 1000 0110 1110 1110 1101 000 100 010 100 010 110 000 110 100 100 00 00 10 00 00 00 10 00 10 00
1069	11111 1001 0111 1110 1110 1101 000 100 010 100 010 110 000 110 100 100 00 00 10 00 00 00 10 00 10 00
1070	11111 1001 0111 1110 1110 1101 000 100 010 101 010 110 000 110 100 100 00 00 10 00 00 00 10 00 10 00
1071	11111 1100 1110 1110 0111 0111 110 110 110 000 010 010 000 010 010 100 11 00 00 00 00 00 00 00 10 00
1072	11111 1001 0111 1110 1110 1101 000 101 010 101 010 110 000 110 100 100 00 00 10 00 00 00 10 00 10 00
1073	11111 1100 1110 1110 1111 1111 110 110 110 000 010 110 000 010 110 100 11 00 00 00 00 00 00 00 10 00

#	Binary representation
	...

Table5.: An example class of the quintary case with 6 sets of functional dependencies

Investigation of the number of cases with more than 5 attributes is an open issue. What is the exact number of valid sets and different types for 6 or 7 attributes? How can the number of sets be determined or estimated for higher number of attributes (lower and upper bounds)? Answering these questions still need future work.

2.8 Summary of the Number of Closed Sets

Let n be the number of attributes of the considered relation schema. Denote by $\mathcal{SD}_n$ the set of closed sets of (singleton, non-trivial) functional dependencies for this n (with constant attributes disallowed). Defining τ as 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 set of different classes is $\mathcal{SD}_n/\tau$. We are focusing on these different classes and the size of this set. Another possibility is to let the attributes to be stated as constants. Performing this extension to $\mathcal{SD}_n$ we get a larger set, denoted by $\mathcal{SD}_n^0$. The different cases (types) of functional dependency sets taking zero-dimensional constraints into account form the set $\mathcal{SD}_n^0/\tau$. It can be easily verified that $\left|\mathcal{SD}_{n+1}^0/\tau\right| = |\mathcal{SD}_{n+1}/\tau| + \left|\mathcal{SD}_n^0/\tau\right|$ holds for each $n \in \mathbb{N}^+$.

With these notations, Table 6 shows the number of closed sets of functional dependencies

n	$\lvert\mathcal{SD}_n\rvert$ (1)	$\lvert\mathcal{SD}_n/\tau\rvert$ (2)	$\lvert\mathcal{SD}_n^0\rvert$ (3)	$\lvert\mathcal{SD}_n^0/\tau\rvert$ (4)
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

Table6. Number of closed sets of functional dependencies for n attributes (1), *number of different types of sets* (with equivalent sets treated as one) (2) and the same values with zero-dimensional constraints ($\emptyset \to A$) allowed (3, 4)

for unary, binary, ternary, quadrary and quintary relational schemata.

2.9 The Impact of Sets of Functional Dependencies

2.9.1 The Cardinality Constraint Notation of Functional Dependencies

2.9.2 Decomposability

Ternary Relationship Types and their Decomposability We notice that case # 10 is the only case which does not allow a BCNF representation of the relationship types. We may classify the case by their decompositions:

No decomposition can be applied: The connectivity in the graphs of the functional dependencies does not allow a decomposition in cases # 0, # 8, # 9, # 10, # 11, # 12, and #13. The reasons are various:

- The components are not associated to each other.
- The components are heavily inter-twinned with each other.
- There does not exist a dependency preserving decomposition (# 10) into a BCNF or a non-redundant decomposition in a 3NF. In case # 10 we would repeat the group A, B again in the other component. We notice that [**?**] proposed in the case to split the component. He considered the `city` (A), `ZIP` (B) and `street` (C) example and proposed to split the ZIP component into two, e.g. ZIP-digits-for-city (B_1) and Additional-ZIP-digits (B_2). In this case we obtain the system $\{B_1 \to A,\ A \to B_1,\ AC \to B_2\}$.

Decomposition is possible but not preferable: Decomposition can be applied and a split into two relationships types is possible. This decomposition is useful, however, only in the case if the split components doe not have any association among them. This pattern of behavior can be observed in cases # 3, # 5, and # 7.

Unique decomposition: The relationship type can be uniquely split into two relationship types. This property is observed in cases # 1, # 2, and # 4.

Several decompositions can be applied: Case # 6 allows to apply two different decompositions. Which one is the most appropriate depends on the application.

We observe, however, that the potential decomposabilty can directly be observed in the figures for the cases # 1, # 2, # 4, and # 6. Therefore, we may use these patterns for reasoning on decompositions.

Chapter 3

The Graphical Representation of Sets of Functional Dependencies

There have been several proposals for graphical representation of sets of functional dependencies. Well-known books such as [AT93] and [Yan86] have used a graph-theoretic notion. Nevertheless, these graphical notations have not made there way into practice and education. The main reason for this failure is the complexity of representation. Graphical representations are simple as long as the set of functional dependencies are not too complex. Especially, if all functional dependencies have a singleton left hand side the graphical representation of functional dependencies allows also reasoning.

[Cam02] has proposed a representation for the ternary case. This representation is simple in the case of ternary relationship types. It is, however, not generalizable to the case of quadrary relationship types. At the same time, it contains details which are unnecessary since the entity-relationship model is based on the assumption that only object of the component classes that are stored in the database can be associated to each other via a relationship type. Due to this, there are dependencies inside the the representation. Therefore, the graphical notation becomes error-prone.

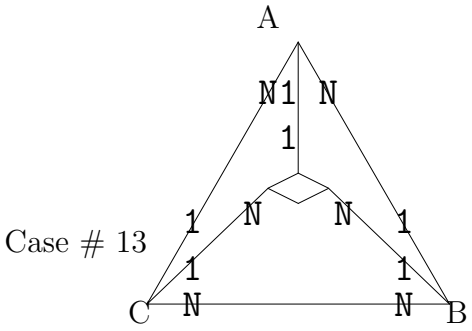

Fig. 1. The notation of functional dependencies in the [Cam02] representation

Therefore, we propose another representation.

3.1 The Ternary Case

The Graphical Representation through Two Triangulars The notations given in [Cam02] are far too complex. We use a simpler notation which reflect the validity of functional depen-

dencies in a simpler and better understandable fashion. We distinguish two kinds of functional dependencies:

One-dimensional (singleton left sides): Functional dependencies of the form $\{A\} \to \{B, C\}$ can be decomposed to canonical functional dependencies $\{A\} \to \{B\}$ and $\{A\} \to \{C\}$. They are represented by endpoints of binary edges in the triangular representation.

Two-dimensional (two-element left sides): Functional dependencies with two-element left-hand sides $\{A, B\} \to \{C\}$ cannot be decomposed. They are represented in the triangular on the node relating their right side to the corner.

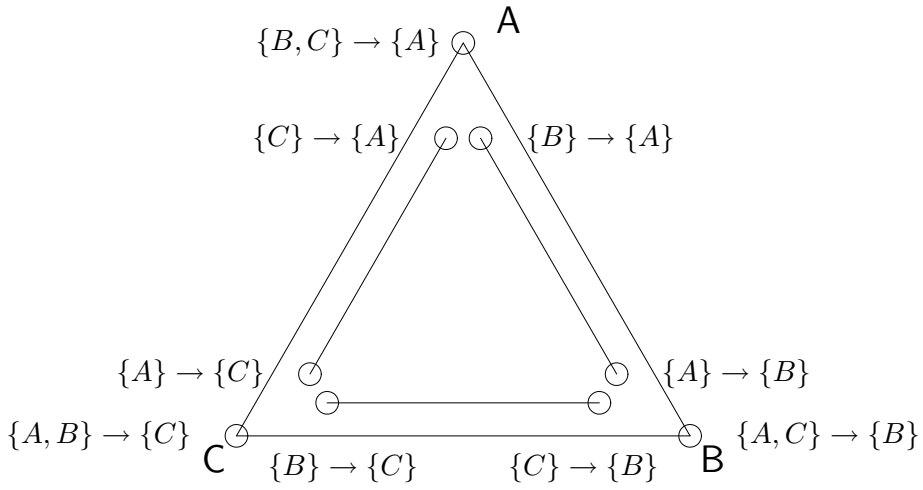

Fig. 2. Triangular representation of sets of functional dependencies for the ternary case

We may represent also candidates for excluded functional dependencies by

crossed circles for the case that we know that the corresponding functional dependency is not valid in applications or by

small circles for the case that we do not know whether the functional dependency holds or does not hold.

This representation in shown on Figure 2. It seems to be quite natural and has the advantage that it can be directly transferred to arbitrary arity. The representation uses an inside-out development by the arity of the left hand side (ie. dimension) of the canonical functional dependency. Figure 3 presents an alternative variant compared to the original one.

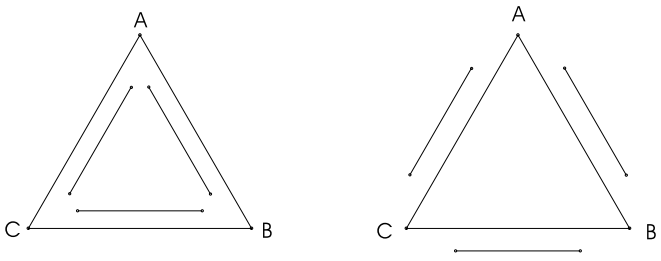

Fig. 3. Comparison of two variants of the triangular representation

We use now the following notations in the figures:

Basic functional dependencies are denoted by filled circles.
Implied functional dependencies are denoted by circles.
Negated basic functional dependencies are either denoted by dots or by crossed filled circles.
Implied negated functional dependencies are either denoted by dots or by crossed circles.

Figure 4 shows some examples of the triangular representation. All different ternary cases of closed sets (relationship types) are shown on Figure 5.

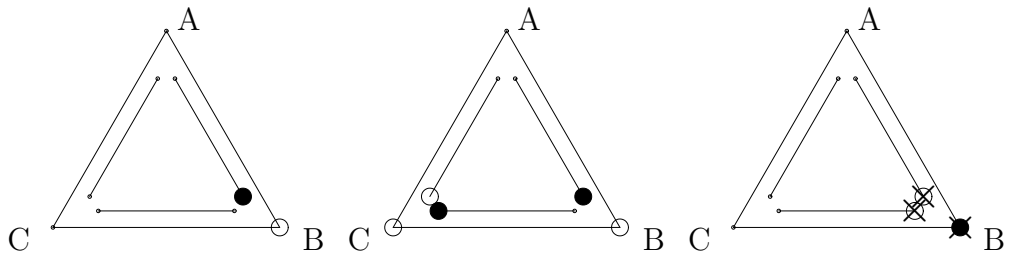

Fig. 4. Examples of the triangular representation. From left to right: 1. The functional dependency $\{A\} \rightarrow \{B\}$ and the implied functional dependency $\{A, C\} \rightarrow \{B\}$. 2. The functional dependencies $\{A\} \rightarrow \{B\}$, $\{B\} \rightarrow \{C\}$ and their implied functional dependencies. 3. The negated functional dependency $\{A, C\} \nrightarrow \{B\}$ and the implied negated functional dependencies $\{A\} \nrightarrow \{B\}$ and $\{C\} \nrightarrow \{B\}$

These cases corresponds to the following cases of cardinality constraint sets:

Case #0 : card(R,X) = (0,n) or card(R,X) = (0,.)
Case #3 : card(R,C) = (0,1)
Case #7 : card(R,A) = (0,1), card(R,B) = (0,1)
Case #8 : card(R,A) = (0,1), card(R,B) = (0,1), card(R,C) = (0,1)
Case #11 : card(R,BC) = (0,1)
Case #12 : card(R,AC) = (0,1), card (R,BC) = (0,1)
Case #13 : card(R,AB) = (0,1), card (R, AC) = (0,1), card(R,BC) = (0,1)

The rest of the cases need an embedded (so-called [Tha00] "projected") cardinality constraint:

Case #1 : card(R[AB],B) = (0,1)
Case #2 : card(R[AB],B) = (0,1), card(R[AC],C) = (0,1)
Case #4 : card(R[AB],B) = (0,1), card(R[BC],C) = (0,1)
Case #5 : card(R[AB],A) = (0,1), card(R[AB],B) = (0,1)
Case #6 : card(R[AB],A) = (0,1), card(R[AB],B) = (0,1), card(R[BC],C) = (0,1)
Case #9 : card(R,AB) = (0,1), card(R[AC],C) = (0,1), card(R[BC],C) = (0,1)
Case #10 : card(R,AC) = (0,1), card(R[AB],B) = (0,1)

3.2 The Quadrary Case

As mentioned above, the triangular representation can be generalized to higher number of attributes. Generalization can be performed in two directions: representation in a higher-dimensional space (3D in the case of 4 attributes) or constructing a planar (2D) representation. The two-dimensional version has usually two subtypes: redundant (with some dependencies

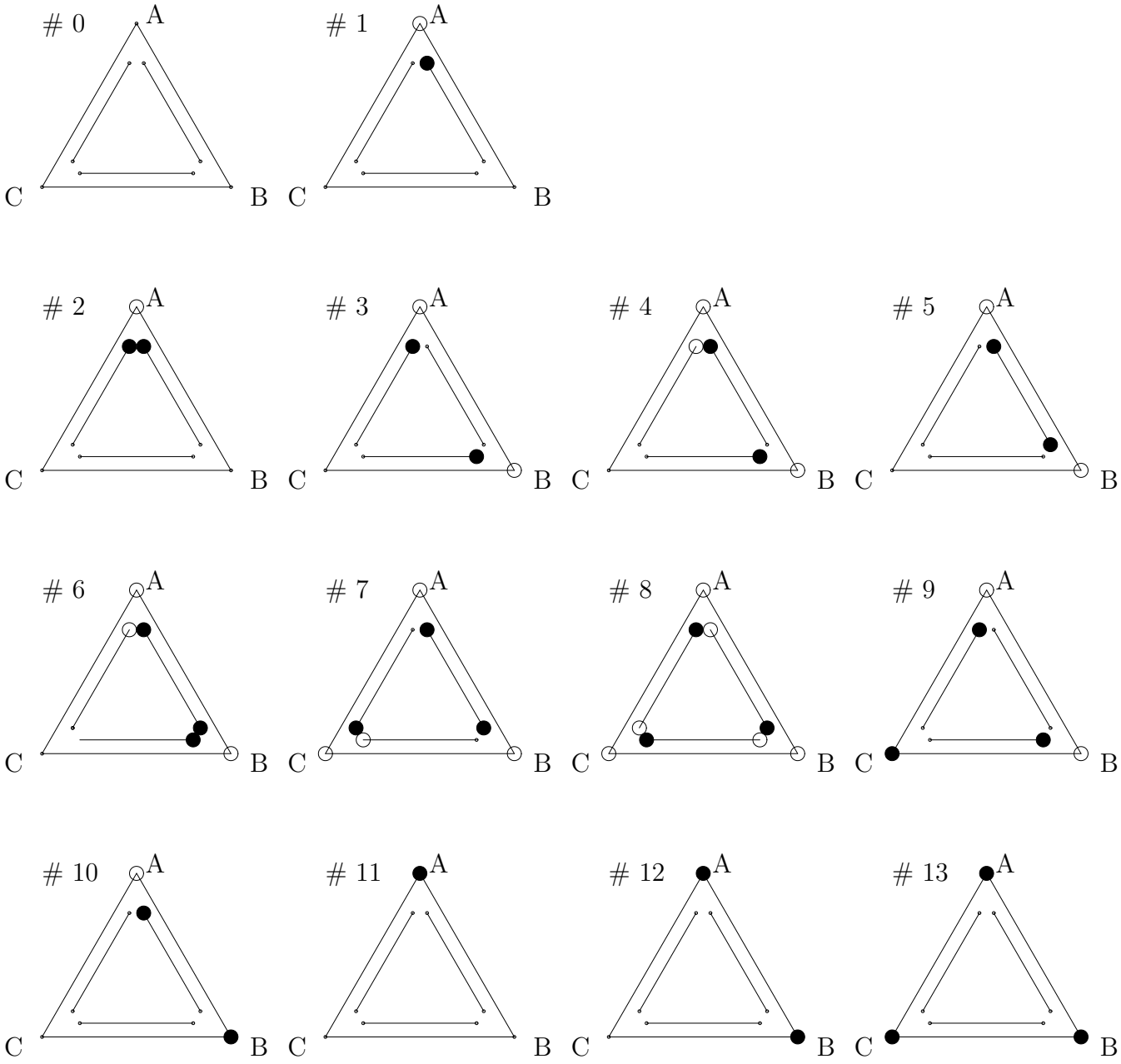

Fig. 5. All sets of functional dependencies in ternary relationship types

represented twice or more times) and non-redundant. We present the higher-dimensional representation first for four attributes (based on a tetrahedron) and the planar representations afterwards.

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 constraint placeholder. One-dimensional dependencies are represented as endpoints of the edges (which are one-dimensional too), while the representation of a two-dimensional dependency is a vertex of the triangle which is a two-dimensional shape. It is straightforward that the quadrary case contains four nested ternary cases with their one-dimensional parts (edges) shared. Additionally, three-dimensional constraints can be represented as verteces of a three-dimensional shape which is actually a tetrahedron. This way we get a representaion in 3D space, where each node

is a placeholder of a functional dependency or excluded constraint (see Figure 6). For better visibility, separate edges are drawn outside the tetrahedron where possible (like the second variant of the triangular representation for the ternary case).

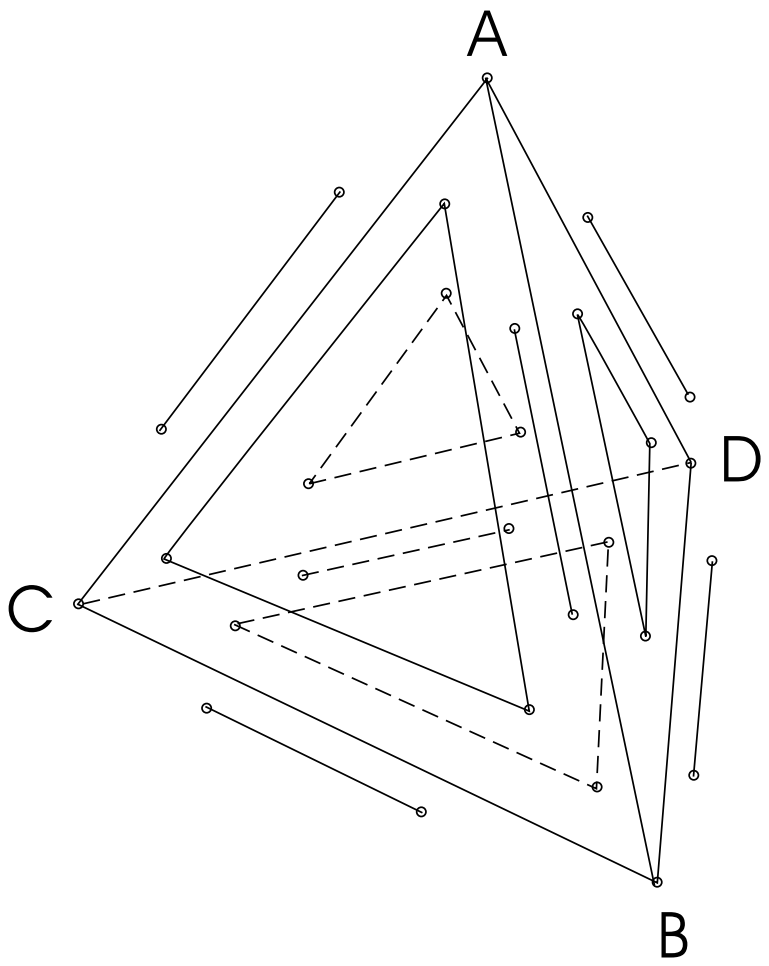

Fig. 6. A tetrahedron as the 3D graphical representation for four attributes (stripped lines indicate invisible edges from the front)

Substituting the 3-dimensional tetrahedron with a square, another version more suitable for 2-dimensional presentation can be constructed with the nested triangulars drawn inside (see Figure 7). The advantage of this type of graph is that the lines do not intersect and the four nested ternary cases can be clearly seen. However, the latter causes the drawback of it since the one-dimensional parts are drawn twice (12 edges of the 4 separate triangles are drawn instead of the 6 shared edges of the tetraherdon). This redundancy can be avoided by using the quadratic representation presented on Figure 8. As an example for both types of quadratic representation, Figure 9 presents the set of functional dependencies generated by $\{B \rightarrow A, AC \rightarrow B\}$. We use the non-redundant quadratic representation together with the tetrahedral representation hereafter.

Section 2.6 presented the different types of sets for four attributes (165 sets), classified according to the dimensions of attributes. The class of ($[A] = [B] = 1, [C] = [D] = 2$) is presented in the graphical form on Figure 10 as an example. We omit the graphical presentation of all cases.

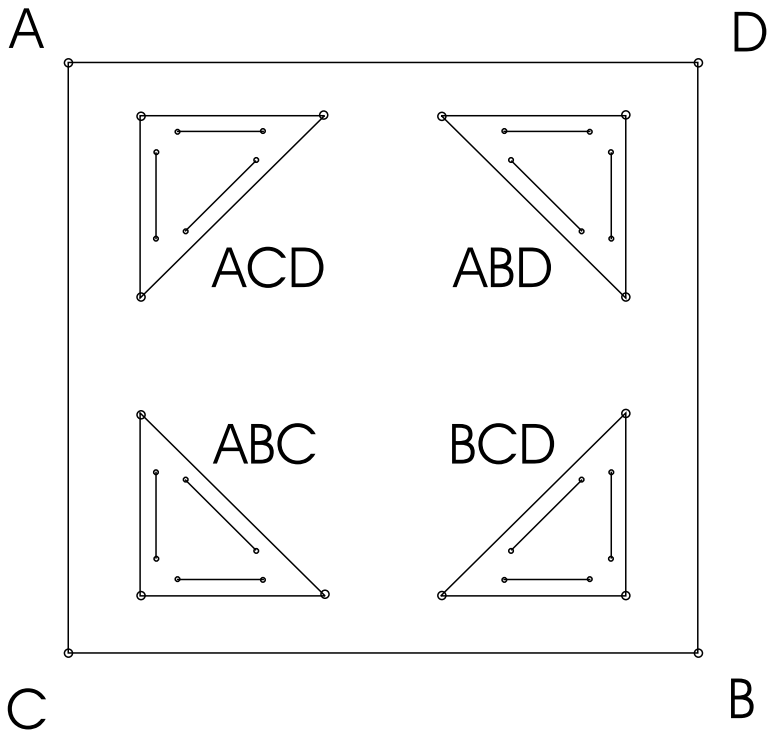

Fig. 7. The redundant quadratic representation for four attributes

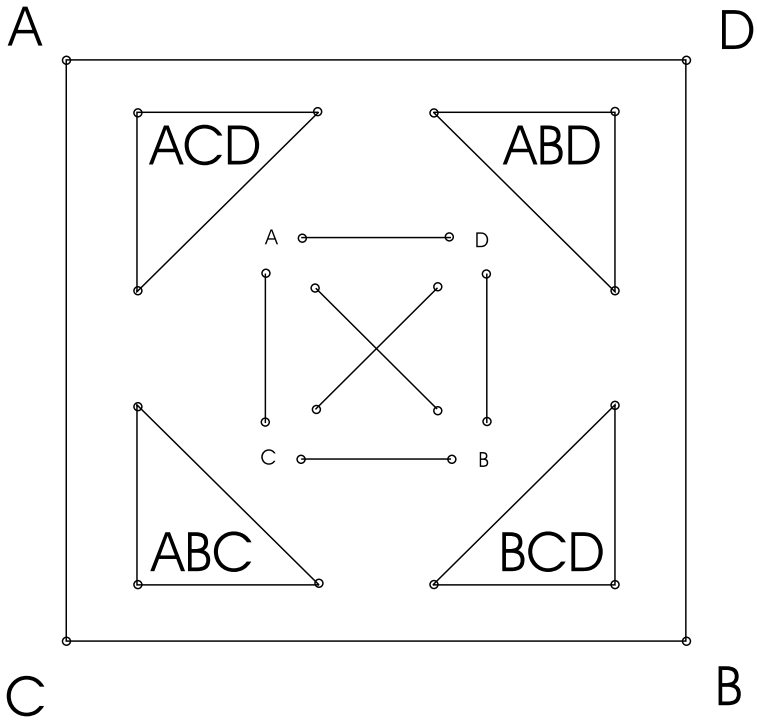

Fig. 8. The quadratic representation without constraint redundancy

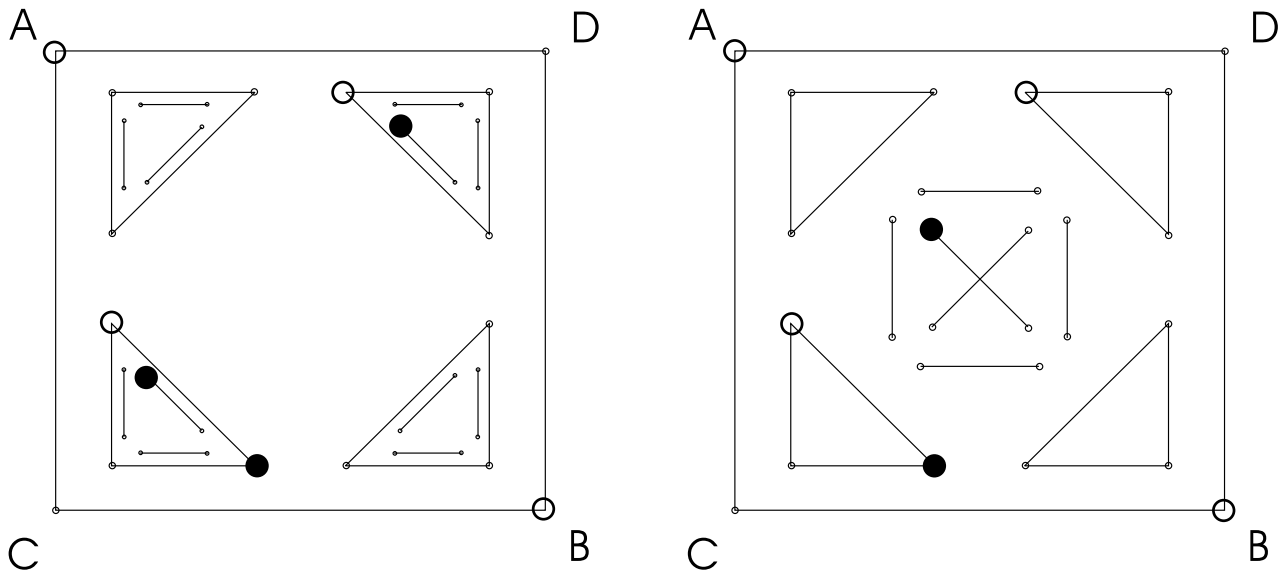

Fig. 9. Comparing the two types of quadratic representation for the set generated by $\{B \to A, AC \to B\}$

3.3 The Quintary Case

The higher-dimensional representation for 5 attributes exists in the 4D space with 5 nested quadrary 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 quadrary cases, while each of the triangles (corresponding to a ternary case) has 2 neighbouring quadrary cases. The frame of the four-dimensional object is shown on Figure 11 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 quardary case and is shown on Figure 12. Each node of the graph is a placeholder of a functional constraint (dependency or negated constraint) placeholder. Note this representation 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 dicover which 3 edges belong to a specific triangle or which 4 triangles belong to a specific trapezoid (or corresponding tetrahedron) by looking at the parallel lines and directions of attributes.

3.4 Possible Generalizations

3.4.1 Incorporating constant attributes

For a complete representation of sets of functional constraints, it is neccessary to incorporate dependencies of the form $\emptyset \to A$ into the system where A is an attribute (and negated constraints $\emptyset \not\to X$ where X is a nonempty set of attributes), although they are usually not needed for data modeling issues. Extending our graphical representation is straightforward, making the geometrical analogy complete by incorporating the verteces themselves as separately drawn

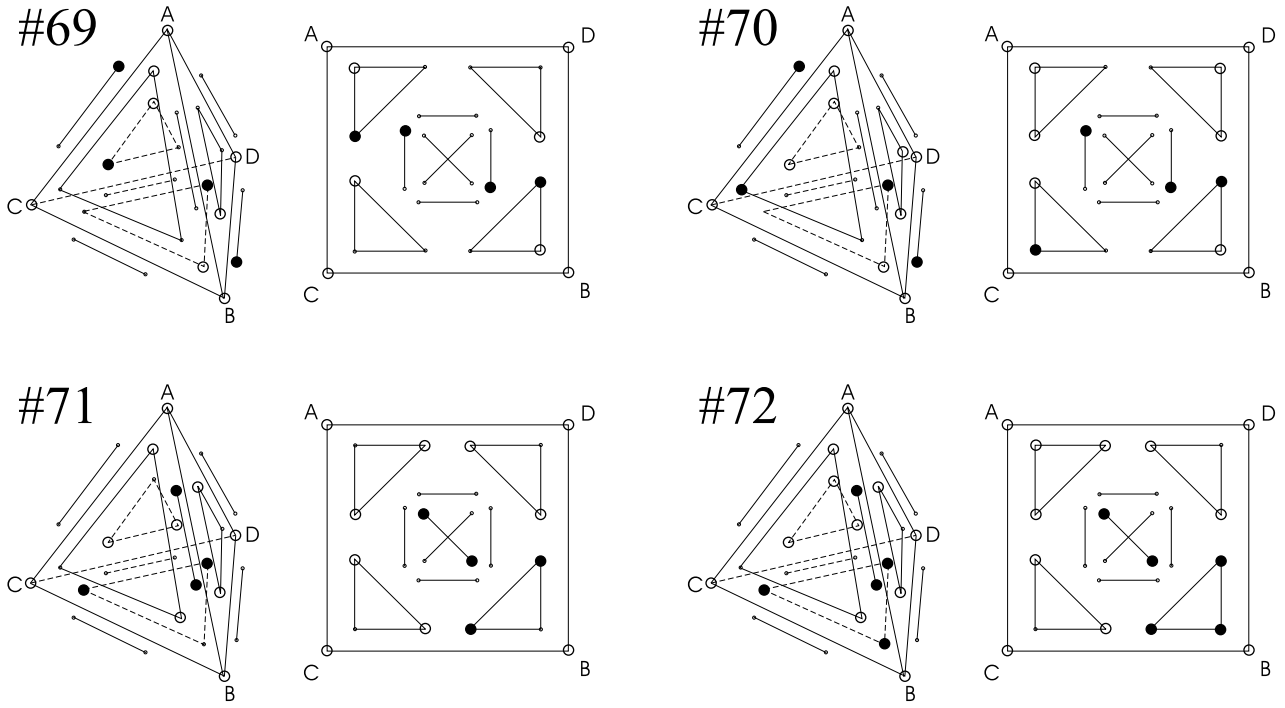

Fig. 10. The tetrahedral and quadratic graphical representations of sets with $[A] = [B] = 1, [C] = [D] = 2$ (numbers of sets refer to the spreadsheet representation presented in Section 2.3)

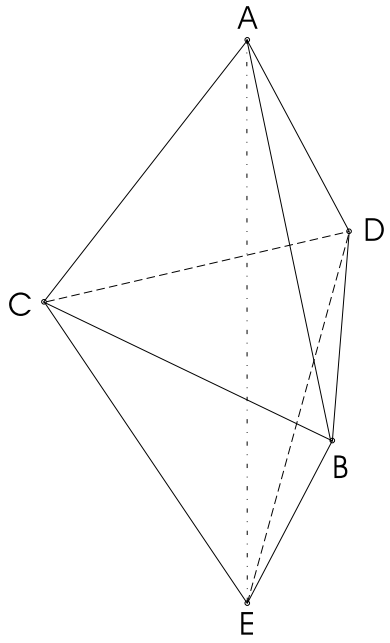

Fig. 11. A three-dimensional projection of the frame of the four-dimensional object for the representation of FD sets over 5 attributes. The five tetrahedra corresponding to nested quadrary cases can easily be discovered, sharing their surface triangles (they represent the ten nested ternary cases) and edges (ten nested binary cases).

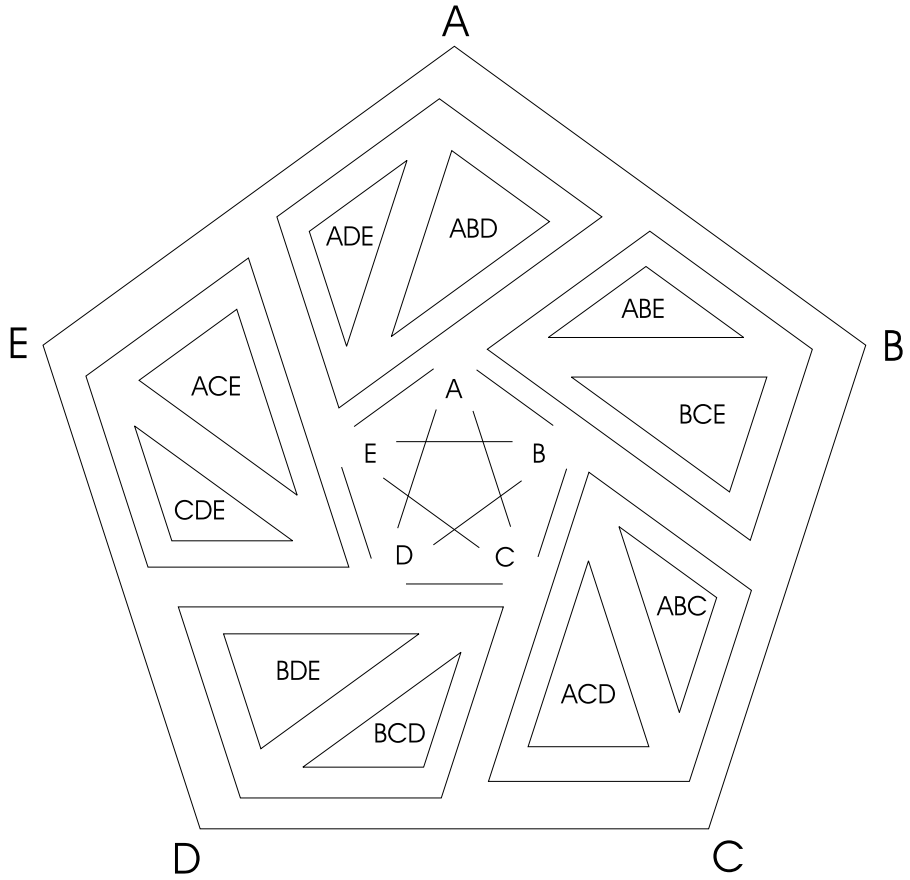

Fig. 12. The pentagonal representation for sets of functional dependencies over 5 attributes

zero-dimensional components and placeholders of zero-dimensional constraints (see Figure 13 for the extended triangular and tetrahedral representations).

3.4.2 Increasing the number of attributes

As we introduced the notion of dimension (Section 2.2), the generalization of the triangular (tetrahedral, etc.) representation towards to a higher number of attributes is theoretically starightforward. For a number of attributes n, a generalized triangular representation can be constructed in the $n-1$ dimensional space[1]. Although these higher-dimension representations may be used as data structures for storing sets in the memory of a computer, their graphical presentation becomes more complicated with the number of attributes raised.

For instance, the generalized triangular representation for 6 attributes exists in the 5D space. A possible way of constructing a non-redundant 2D representation is shown on Figure 14. A case over six attributes has 6 nested quintary cases (4D-objects or pentagons), 15 nested quadrary cases (tetrahedra or quadrangles), 20 ternary and 15 binary cases (triangles and edges, respectively). The total number of constraint placeholders (nodes) is 186.

If we replace the notion of attributes with entites, this also illustrates the contrast between a complete description of a relationship with higher arity and the usual notation of an entity-relationship graph. Certainly, most of the relationship types (sets of functional dependencies) occur rarely in practice and relationships with higher arity are usually decomposed.

[1] Dimension of a constraint is at most $n-1$. This range was extended with ∞ for the attributes for indicating the attribute is not dependent (its dimension is out of range, it would be n-dimensional if trivial constraints were allowed).

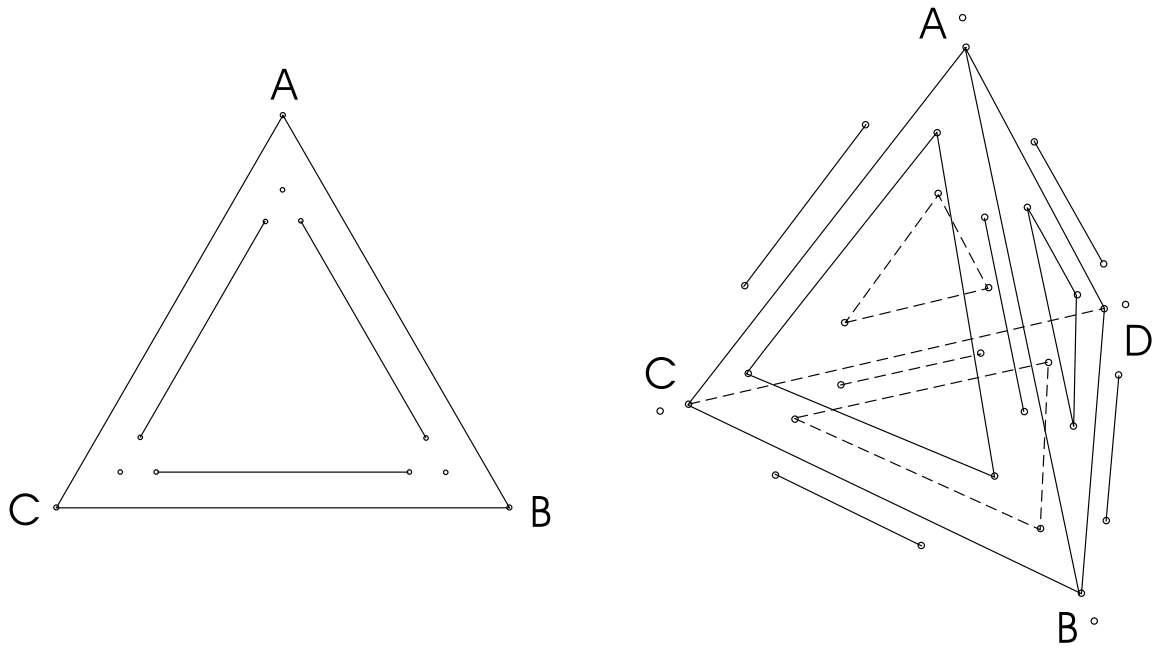

Fig. 13. Extending the triangular and tetrahedral representations with placeholders of zero-dimensional constraints as separate vertex nodes

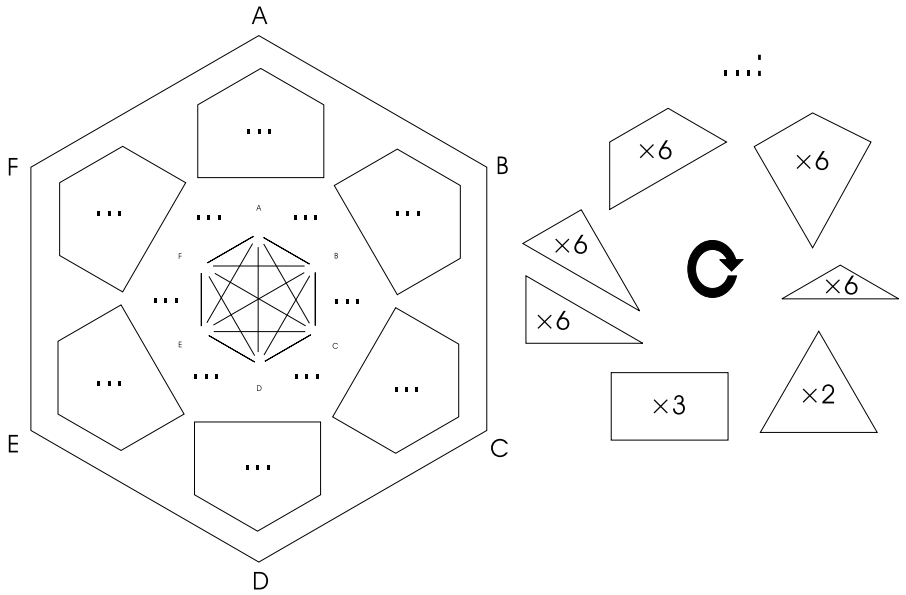

Fig. 14. Towards the hexagonal representation as a possible generalization for 6 attributes. The frame on the left-hand side is to be extended by rotations of components on the right-hand side

Detailed investigation of cases with 6 or more attributes is still an open issue.

Chapter 4

Implication Systems for the Graphical and Spreadsheet Representations

The Extended Armstrong Implication System Excluded functional constraints and functional dependencies are axiomatizable by the following formal system [Tha00].

Axioms

$$XY \;\rightarrow\; Y$$

Rules

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

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

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

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

Rules (3) and (7) are one of the possible inversions of rule (2) since the implication $\alpha \wedge \beta \;\rightarrow\; \gamma$ is equivalent to the implication $\neg\gamma \wedge \beta \;\rightarrow\; \neg\alpha$. Rules (4) and (5) are inversions of rule (1). Rule (6) can be considered to be the inversion of the following union rule valid for functional dependencies:

$$(8)\ \frac{X \;\longrightarrow\; Y\,,\quad X \;\longrightarrow\; Z}{X \;\longrightarrow\; YZ}$$

This rule can be derived from the axiom and rule (2).

Towards a More Suitable Axiomatization The universe of the extended Armstrong implications system is $\mathbb{D} \cup \mathbb{E}$ (functional dependencies and excluded functional constraints over a set of attributes) while our graphical and spreadsheet representations deal with sets of constraints over $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$ (singleton, non-trivial constraints) which is proper as briefly discussed (see Section 2.1). From the axiomatic point of view, this reduction can be achieved by two steps.

A natural simplification to handle sets of constraints without loosing any relevant information is omitting trivial constraints and non-singleton functional dependencies since trivial dependecies always hold (trivial excluded constraints must always be false, consequently) and

non-canonical functional dependencies can always be substituted by canonical dependecies[1]. However, the axiom and rules of the extended Armstrong implication system do not correspond to this restriction. It will be shown that an equivalent implication system can be constructed if these restrictions are applied to the universe of constraints (systems ST and NST in Section 4). It turns out that no trivial dependencies are needed in the new system to deduce all non-trivial consequences of a given set of (positive and/or negative) constraints. That is the reason we can "forget" about trivial dependencies and trivial negated constraints and completely exclude them from the formal system.

The second step is omitting the non-singleton excluded functional constraints. Although this is a real restriction, non-singleton excluded functional constraints are not needed for describing a closed set of functional dependencies since they are interpreted as disjunctions.[2] For a closed set, it is exactly known which of them holds. Deriving the full knowledge starting with an initial set of constraints may result some non-singleton excluded constraints. However, their relevance is rather low if they can not be simplified.[3] Neither is it likely for an initial set to contain this kind of excluded constraints. It will be shown by constructing another formal system that the non-singleton excluded constraints are not needed for deriving a singleton excluded constraint when the initial set contains only singleton constraints.

The first step mentioned above can be formalized as restricting the universe to $\mathbb{D}_c^+ \cup \mathbb{E}^+$ and for each finite subset $\mathcal{F}$ of $\mathbb{D} \cup \mathbb{E}$ the set $\mathcal{F}' = \{V \rightarrow C \mid \exists W : V \rightarrow W \in \mathcal{F}, C \in W \setminus V\} \cup \{V \nrightarrow U \mid \exists W : V \nrightarrow W \in \mathcal{F}, U = W \setminus V\}$ is an equivalent representation over $\mathbb{D}_c^+ \cup \mathbb{E}^+$. The second step is taking $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$ as the universe and disallowing constraints in the form $V \nrightarrow W$, $|W| > 1$ to be elements of the initial set.

However, the extended Armstrong implication system makes it possible to derive constraints outside $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$ even if the initial set contains only constraints over this universe, and it must be guaranteed these intermediate results are not neccessary for deducing some of the consequences belonging to $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$. This is needed to provide complete implication systems for our representations. Although this might be shown for the extended Armstrong system as well, we present an alternative axiomatization called PQRST Implication System, which is more suitable for the use with our representations. The rules will also have the advantage that each of them refers to only one set of attributes and the rest is of single, different attributes not occuring in the set. This way it becomes much easier to recognize a possibility to apply one of the rules in an attribute-based representation (as opposed to a subset-based representation).

It also turns out there exists a specific order of application of the rules, which leads to a complete deduction algorithm for elicitation of the full knowlegde a set of constraints holds.

---

[1] For example, $X \rightarrow AB$ is represented as $X \rightarrow A$ and $X \rightarrow B$. Excluded functoinal constraints with more than one attribute on their right-hand sides can not be eliminated this way. However, we show that omitting these can also be achieved.

[2] For instace, $X \nrightarrow AB$ means either $X \nrightarrow A$ or $X \nrightarrow B$ (or both) must hold.

[3] If neither $X \rightarrow A$ nor $X \rightarrow B$ can be deduced then $X \nrightarrow AB$ can not be simplified and one just conclude the lack of information: it is not possible to decide whether the dependencies hold or not. Using the closed world assumption as a possible model, one might finally conclude neither of the two dependencies hold. Otherwise, we state the lack of information and as the state of one of the dependencies is determined, a new reasoning session starts with the extended initial set so the non-singleton negated constraint will not be needed.

4.1 The ST and PQRST Implication Systems for the Use with our Representations

In each of the following rules and axioms, letter Y denotes a set of attributes (allowed to be empty), while A, B and C stand for different single attributes not occuring in Y.

(S) $\frac{Y \rightarrow B}{YC \rightarrow B}$ (T) $\frac{Y \rightarrow A, YA \rightarrow B}{Y \rightarrow B}$

(P) $\frac{YC \nrightarrow B}{Y \nrightarrow B}$ (Q) $\frac{Y \rightarrow A, Y \nrightarrow B}{YA \nrightarrow B}$

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

The rules presented here can directly be applied for deducing consequences of a set of constraints given in terms of the graphical or spreadsheet representation, as it will be demonstrated in Section 5.1 and Section 6.

To get implied functional dependencies, rules (S) and (T) can be used. No explicit rule for transititity is needed, Section 5.1 will show some examples on how transitivity can be simulated with these rules. For deducing excluded functional constraints, rules (P), (Q) and (R) act as negations of (S) and (T). ($\Box$) is a formalization of '$\nrightarrow$' being the negation of '$\rightarrow$', ie. $\neg(\Box)$ can be deduced starting with contradictory sets of constraints.

Note we are not dealing with trivial or non-singleton constraints. Restriction of the universe to singleton, non-trivial constraints is possible from the syntactic point of view because none of the implication rules presented above can be used to deduce trivial or non-singleton functional dependencies if the initial set contains only non-trivial and singleton dependencies. The same holds for excluded functional constraints.

With the rules just presented, we define two formal systems for the use with our graphical and spreadsheet representations. One of them deals with non-trivial canonical (singleton) functional dependencies, while the other one extends to their negated forms. The two systems are

- the *ST implication system* over $\mathbb{D}_c^+$ with rules (S) and (T) and no axioms,
- the *PQRST implication system* over $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$with all the presented rules and the symbolic axiom ($\Box$), which is used for indicating contradiction.

These systems are sound and complete for deducing non-trivial, singleton constraints. We put this statement together with others mentioned above into the form of two theorems.

Theorem 1 The ST system is sound and complete over $\mathbb{D}_c^+$, ie. $\mathcal{F} \vdash_{ST} \delta \iff \mathcal{F} \vDash \delta$ for each finite subset $\mathcal{F}$ of $\mathbb{D}_c^+$ and $\delta \in \mathbb{D}_c^+$.

Theorem 2 Let $\mathcal{F}$ be a finite subset of $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$ and $\delta \in \mathbb{D}_c^+ \cup \mathbb{E}_c^+$.
The PQRST system without ($\Box$) is sound over $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$ and complete with the restriction that $\mathcal{F}$ cannot be contradictory, ie. $\mathcal{F} \vdash_{PQRST} \delta \iff \mathcal{F} \vDash \delta$ for each non-contradictory $\mathcal{F}$. Moreover, $\neg(\Box)$ can be derived iff $\mathcal{F}$ is contradictory.[4]

[4] Contradictory means two opposite constraints (e.g. $X \rightarrow W$, $X \nrightarrow W$) can be derived using the extended Armstrong implication system (they might be trivial as well). Showing that $\neg(\Box)$ (with non-trivial, singleton dependencies) can be deduced using PQRST is satisfiable in such cases. Contradictory cases in the Armstrong system allow deduction of some non-trivial negated constraints which can not be derived using PQRST the same way because trivial constraints are not allowed (eg. one can get $X \nrightarrow \emptyset$ with the extended Armstrong system using rule (6) first and then for any B, $X \nrightarrow B$ follows using rule (4)). Although these cases are irrelevant, formal completeness can not be stated rigorously.

4.2 Order of Application of Rules

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.

For positive dependencies, using (S) first as many times as possible and using (T) as many times as possible afterwards is a complete method for geting all the non-trivial positive consequences of a given set of constraints (see Part 1 of Theorem 3 below). This fact was used to count the different sets for the ternary, quardary and quintary cases (see Section 2.4) and this is the way eg. transitivity can be simulated as well (Section 5.1).

A complete method can be achieved (Part 2 of Theorem 3) by using the rules as many times as possible in the following order: (S), (T), (R), (P), (Q) (the order of (P) and (Q) is arbitrary).

Theorem 3 1. Let $\mathcal{F}$ and $\mathcal{G}$ be finite subsets of $\mathbb{D}_c^+$. If $\mathcal{F} \vdash_{ST} \mathcal{G}$ then all elements of $\mathcal{G}$ can be deduced starting with $\mathcal{F}$ by using the rules (S) and (T) the way that no application of (T) precede any application of (S).

2. If $\mathcal{F}$ and $\mathcal{G}$ are finite subsets of $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$ and $\mathcal{F} \vdash_{PQRST} \mathcal{G}$ then all elements of $\mathcal{G}$ can be deduced starting with $\mathcal{F}$ by using the rules (S), (T), (R), (P) and (Q) the way that no application of (T) precede any application of (S), no application of (R) precede any application of (T) and no application of (P) or (Q) precede any application of (R). Order of (P) and (Q) is arbitrary. Furthermore, (R) is needed to be applied at most once if $|\mathcal{G}| = 1$.

4.3 Implication Systems with Non-Singleton Excluded Constraints Allowed

Proofs of Theorems 1 and 2 are based on the soundness and completeness of the Armstrong and extended Armstrong implication systems over $\mathbb{D}$ and $\mathbb{D} \cup \mathbb{E}$, respectively (proofs will follow in Section 4.5). However, rules (S), (T), (P), (Q) and (R) just presented (systems ST and PQRST) can be used for implications over $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$. As an intermediate step, we extend the universe with non-singleton negated constraints ($\mathbb{D}_c^+ \cup \mathbb{E}^+$) and introuce another system called NST in this subsection. It will be shown that the NST system is equivalent to the system PQRST over the original, restricted universe ($\mathbb{D}_c^+ \cup \mathbb{E}_c^+$). We also introduce two other systems called U and UE afterwards, which are equivalent to the ST and NST systems, respectively. Systems U and UE will then be compared to the (extended) Armstrong axiomatization which leads to the completion of the proofs of theorems.

The universe is now extended to $\mathbb{D}_c^+ \cup \mathbb{E}^+$, ie. non-singleton excluded functional constraints are allowed.

The NST Implication System as an Extension to the PQRST System In each of the following rules and axioms, letters Y, Z and W denote pairwise disjoint sets of attributes (the sets are allowed to be empty with the restriction that right-hand sides of excluded constraints can not be empty), while A, B and C stand for different single attributes not occuring in the sets of the same rule.

(S) $\frac{Y \rightarrow B}{YC \rightarrow B}$ (T) $\frac{Y \rightarrow A, YA \rightarrow B}{Y \rightarrow B}$

(NS) $\frac{YC \nrightarrow Z}{Y \nrightarrow Z}$ (NT1) $\frac{Y \rightarrow A, Y \nrightarrow Z}{YA \nrightarrow Z}$ $(*)\frac{Y \nrightarrow Z}{Y \nrightarrow ZC}$

(NT2) $\frac{YZW \rightarrow B, Y \nrightarrow ZB}{Y \nrightarrow ZW}$ $(\Box)$ $\neg(Y \rightarrow B,\ Y \nrightarrow B)$

(NT2S) $\frac{Y \rightarrow B, Y \nrightarrow ZB}{Y \nrightarrow Z}$

For deriving functional dependencies (positive constraints), rules ((S) and (T)) are the same as before. For deducing excluded functional constraints, rules (NS), (NT1), (NT2) and $(*)$ act as negations of (S) and (T) now. Although (NT2S) can be derived from (S) and (NT2) with $W = \emptyset$ (see Lemma 8 later), it is inlcuded separately since this case may often arise.

In practice, $(*)$ is not needed since all relevant information for excluded constraints can be derived without using it, as it will be shown (see Theorem 4, part 2). It means we do not need to extend the right-hand sides of the negated constraints during elictication of the full knowledge on valid constraints. This is important since negated dependencies with more than one attribute on their right-hand sides are interpreted as disjuctions and are not represented directly in the graph or spreadsheet and our goal is usually to reduce their right-hand sides. Using (NT2) remains the only case that can increase the size of the right-hand side of a negated constraint but with removing at least one attribute B at the same time ($B \notin W$).

We define the *NST implication system* over $\mathbb{D}_c^+ \cup \mathbb{E}^+$as all the presented rules ((NT2S) is optional) included and the symbolic axiom $(\Box)$ which is used for indicating contradiction. This system is sound and complete for deducing non-trivial constraints. Denote by *NST'* the formal system NST with $(*)$ excluded.

Theorem 4 Let $\mathcal{F}$ be a finite subset of $\mathbb{D}_c^+ \cup \mathbb{E}^+$ and $\delta \in \mathbb{D}_c^+ \cup \mathbb{E}^+$.

1. The NST system without $(\Box)$ is sound over $\mathbb{D}_c^+ \cup \mathbb{E}^+$and complete with the restriction that $\mathcal{F}$ cannot be contradictory, ie. $\mathcal{F} \vdash_{NST} \delta \iff \mathcal{F} \Vdash \delta$ for each non-contradictory $\mathcal{F}$. Moreover, $\neg(\Box)$ can be derived iff $\mathcal{F}$ is contradictory.
2. Using rule $(*)$ is not neccessary for deducing all relevant information with NST, ie. the NST' system is still sound over $\mathbb{D}_c^+ \cup \mathbb{E}^+$, complete over $\mathbb{D}_c^+$ and if $\mathcal{F}$ is not contradictory and $\delta \in \mathbb{E}^+$, $\delta = X \nrightarrow Y$, then $\mathcal{F} \Vdash \delta \Longrightarrow \exists Z \subseteq Y, Y \neq \emptyset : \mathcal{F} \vdash_{NST'} X \nrightarrow Z$.[5] If $\mathcal{F}$ is contradictory then $\mathcal{F} \vdash_{NST'} \neg(\Box)$ holds.

Proof of Theorem 2 is based on Theorem 4 and the following lemmas stating the equivalence of systems PQRST and NST over $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$.

Lemma 1

If NST is sound then PQRST is sound, ie. $\vdash_{NST}\{(P),(Q),(R)\}$.

Lemma 2

Starting with a set of non-trivial, singleton (positive or negative) constraints each non-trivial, singleton constraint that can be deduced using the NST' system over $\mathbb{D}_c^+ \cup \mathbb{E}^+$ can also be deduced using the PQRST system over $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$ if the initial set is not contradictory. For a contradictory set over $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$, $\neg(\Box)$ can be deduced using PQRST. More precisely,

[5] Note that $X \nrightarrow Z$ holds at least the same or even more information than δ.

– let $\mathcal{F}$ be a finite subset of $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$, X a set of attributes and A a single attribute such that $A \notin X$. If $\mathcal{F}$ is not contradictory and $\mathcal{F} \vdash_{NST'} X \twoheadrightarrow A$, then $\mathcal{F} \vdash_{PQRST} X \twoheadrightarrow A$. For a contradictory set $\mathcal{F}$, $\mathcal{F} \vdash_{PQRST} \neg(\square)$ holds.
– Moreover, if $\mathcal{F} \vdash_{PQRST} X \twoheadrightarrow A$ holds, then deduction can be performed so that rule (R) is used at most once.

4.4 The U and UE Implication Systems and their Relationship with the ST, NST and Extended Armstrong Implication Systems

Consider the following rules. Like before, letters X, Y, Z and W denote pairwise disjoint sets of attributes for each rule they occur in (they are allowed to be empty with the restriction that right-hand sides of excluded constraints can not be empty) while A_i's ($i \in [1..k]$, $k \in \mathbb{N}_0$) refer to distinct attributes not occuring in any of the attribute sets of the same rule.

$$\text{(U)}\ \frac{XY \to A_1, \cdots, XY \to A_k, YA_1 \cdots A_k \to B}{XY \to B}$$
$$\text{(E1)}\ \frac{XY \to A_1, \cdots, XY \to A_k, XY \twoheadrightarrow ZA_1 \cdots A_l\ (0 \le l \le k)}{YA_1 \cdots A_k \twoheadrightarrow Z}$$
$$\text{(E2)}\ \frac{X \to A_1, \cdots, X \to A_k, X \twoheadrightarrow ZA_1 \cdots A_k}{X \twoheadrightarrow Z}$$
$$\text{(E3)}\ \frac{XZW \to A_1, \cdots, XZW \to A_k, XY \twoheadrightarrow ZA_1 \cdots A_k}{XY \twoheadrightarrow ZW}$$
$$\text{(E}\square\text{)}\ \neg(X \to A_1, \cdots, X \to A_k, X \twoheadrightarrow A_1 \cdots A_k)$$

Let us call the system of $\{\text{(U)}\}$ over $\mathbb{D}_c^+$ *U implication system*. Extended by (E1), (E2), (E3) and (E$\square$) we get the *UE implication system* for $\mathbb{D}_C^+ \cup \mathbb{E}^+$.

Note that k (and l) can be 0 for each of the rules except (E2) (irrelevant case). It will be shown that all relevant information regarding the constraints hold at a particular case can be deduced without applying (E3) for $k = 0$ (this case corresponds to $(*)$ in the NST system). This statement as well as the equivalence bewteen UE and the extended Armstrong implication system over $\mathbb{D}_C^+ \cup \mathbb{E}^+$ are formalized by the following lemmas. We use the notation $\vdash_A$ for provability using the Armstrong system for functional dependencies (the axiom and the rules (1) and (2) presented in section 4, over $\mathbb{D}$) and $\vdash_{EA}$ for provability using the extended Armstrong system (over $\mathbb{D} \cup \mathbb{E}$, extended with the rest of the rules). These systems are known to be sound and complete.

Lemma 3 (U) is sound, ie. $\vdash_A$(U).

Lemma 4 System U is complete for functional dependencies with the restriction that trivial dependencies might not be proved and dependencies with more than one attribute on their right-hand sides are represented as canonical dependencies. More precisely, let $\mathcal{F}$ be a finite subset of $\mathbb{D}$. Then, for each two sets of attributes X and Z, $\mathcal{F} \vdash_A X \to Z \Longrightarrow \forall B \in Z \setminus X$: $\mathcal{F}' \vdash_U X \to B$ where $\mathcal{F}' = \{V \to C \mid \exists W : V \to W \in \mathcal{F}, C \in W \setminus V\}$.

Lemma 5 (E1), (E2), (E3) and (E$\square$) are sound, ie. $\vdash_{EA}\{\text{(E1), (E2), (E3)}\}$ and if $\mathcal{F} \vdash_{UE} \neg$(E$\square$) for a finite set $\mathcal{F} \subset \mathbb{D}_c^+ \cup \mathbb{E}^+$, then $\mathcal{F}$ is contradictory.

Lemma 6 System UE is complete for excluded functional constraints when the initial set of constraints is not contradictory and with the restriction that trivial constraints might not be proved. Positive constraints with more than one attribute on their right-hand sides are represented by canonical constraints (similarly to Lemma 4) and each non-self-contradictory trivial negated constraint is represented as a non-trivial negated constraint. More precisely,

- let $\mathcal{F}$ be a finite, non-contradictory subset of $\mathbb{D} \cup \mathbb{E}$ and let X and Z be two sets of attributes such that $Z \setminus X \neq \emptyset$. If $\mathcal{F} \vdash_{EA} X \twoheadrightarrow Z$, then $\mathcal{F}' \vdash_{UE} X \twoheadrightarrow Z \setminus X$, where $\mathcal{F}' = \{V \to C \mid \exists W : V \to W \in \mathcal{F}, C \in W \setminus V\} \cup \{V \twoheadrightarrow U \mid \exists W : V \twoheadrightarrow W \in \mathcal{F}, U = W \setminus V\}$.
- Additionally, if $\mathcal{F}' \vdash_{UE} X \twoheadrightarrow V$ for a non-trivial negated constraint (and $\mathcal{F}'$ is not contradictory), then $\exists U \subseteq V, U \neq \emptyset$ such that $X \twoheadrightarrow U$ can be deduced in the UE system without using (E3) for $k = 0$.
- If $\mathcal{F}$ is contradictory and the corresponding $\mathcal{F}' \subset \mathbb{D}_c^+ \cup \mathbb{E}^+$ (ie. $\mathcal{F}$ contains no trivial excluded constraints and so $\mathcal{F}'$ contains no constraints in the form $X \twoheadrightarrow \emptyset$), then $\mathcal{F}' \vdash_{UE} \neg(\mathrm{E}\Box)$, and deduction can be performed without using (E3) with $k = 0$.

Equivalence between systems ST and U, as well as between NST and UE is stated by the following lemmas:

Lemma 7 The implication systems ST and U are equivalent over $\mathbb{D}_c^+$, ie. $\vdash_U \{(\mathrm{S}), (\mathrm{T})\}$ and $\vdash_{ST} (\mathrm{U})$.

Lemma 8 The implication systems NST and UE are equivalent over $\mathbb{D}_c^+ \cup \mathbb{E}^+$, ie. additionally to Lemma 7,

- $\vdash_{UE} \{(\mathrm{NS}), (\mathrm{NT1}), (\mathrm{NT2}), (*)\}$,
- $\{(\mathrm{S}), (\mathrm{NT2})\} \vdash_{NST} (\mathrm{NT2S})$, $\neg(\mathrm{E}\Box) \vdash_{NST} \neg(\Box)$,
- $\vdash_{NST} \{(\mathrm{E1}), (\mathrm{E2}), (\mathrm{E3})\}$, $\neg(\Box) \vdash_{UE} \neg(\mathrm{E}\Box)$.
- Moreover, $(*)$ is only needed to deduce (E3) when $k = 0$ and vica versa (neither for deducing the other rules nor for deducing (E$\Box$) from ($\Box$) and vica versa).

4.5 Proofs

Now we prove Lemmas 3–8, followed by 1 and 2. Proofs of Theorems 1, 4, 2 and 3 will follow afterwards, respectively. The order is based on dependencies between lemmas and theorems.

Proof of Lemma 3

For $k = 0$ the result is obvious by using (1) for $V = \emptyset$. Assume now that $k > 0$ and so $XY \to A_1, \cdots, XY \to A_k$ and $YA_1 \cdots A_k \to B$ hold. The axiom states $XY \to Y$. Applying rule (8) several times we get $XY \to YA_1 \cdots A_k$ and using rule (2) afterwards results $XY \to B$. □

Proof of Lemma 4

Let $\mathcal{F} \vdash_A X \to Z$. We show $\mathcal{F}' \vdash_U \mathcal{G} = \{X \to B \mid B \in Z \setminus X\}$ by constructing a proof for that in system U, parallel with the proof of $X \to Z$ in the Armstrong system. Suppose the sequence $\langle f_1, \cdots, f_m \rangle$ is a proof of $X \to Z$. Construction of the new proof is made by ensuring for each item $f_i = V \to W$, elements of the set $\mathcal{F}'_i = \{V \to C \mid C \in W \setminus V\}$ already occur in the new proof before f_{i+1} is considered, ie. for each i there exists a prefix $\langle f'_1, \cdots, f'_j \rangle$ in the new proof so that $\bigcup_{p=1}^{i} \mathcal{F}'_p = \bigcup_{q=1}^{j} \{f'_q\}$. For a particular i, this property is denoted by $\mathcal{P}_i$. Note that this set is a subset of $\mathbb{D}_c^+$ so the proof being constructed is valid in system U over $\mathbb{D}_c^+$ if we use rule (U) only. We start with the empty sequence as a prefix when no f_i's have been considered yet.

Let $0 < i \leq m$ and suppose by induction $\langle f'_1, \cdots, f'_j \rangle$ is the prefix of the new proof already constructed according to $\langle f_1, \cdots, f_i \rangle$ with the property just mentioned ($\mathcal{P}_i$). We perform an

(optional) extension resulting $\langle f'_1, \cdots, f'_j, f'_{j+1}, ..., f'_{j+h}\rangle$ so that $\mathcal{F}'_{i+1} \subseteq \bigcup_{q=1}^{j+h}\{f'_q\}$, resulting that $\mathcal{P}_{i+1}$ holds. This will complete the proof since $\mathcal{F}'_m = \mathcal{G}$ because $f_m = X \to Z$ and m is reached by induction.

Let $f_{i+1} \in \mathcal{F}$. In this case, we simply add the elements of the set $\mathcal{F}'_{i+1}$ to the new proof. This step is valid since $\mathcal{F}'_i \subseteq \mathcal{F}'$.

If f_{i+1} is an instance of axiom $XY \to Y$ then $F'_{i+1} = \emptyset$ and nothing is to be done to ensure $\mathcal{P}_{i+1}$.

Let $f_{i+1} = X'V'W' \to Y'V'$ be a result of the application of rule (1) to a previous $f_{i'} = X' \to Y'$. This case, the set $\mathcal{F}'_{i+1} = \{X'V'W' \to D \mid D \in Y'V' \setminus X'V'W'\}$ must be added to the prefix of the new proof if not empty. For a particular element $X'V'W' \to D$ of $\mathcal{F}'_{i+1}$, $D \in Y'V' \setminus X'V'W' = Y' \setminus X'V'W' \subseteq Y' \setminus X'$ and so $X' \to D \in \mathcal{F}'_{i'}$, already occuring in the new proof. Applying rule (U) with $k = 0$, we get $X'V'W' \to D$ and we extend the new proof by this dependency. This step can be performed for each element of $\mathcal{F}'_{i+1}$, resulting $\mathcal{P}_{i+1}$ is true.

The remaining case is when $f_{i+1} = X' \to Z'$ is a result of the application of rule (2) to previous items $f_{i_1} = X' \to Y'$ and $f_{i_2} = Y' \to Z'$. If $\mathcal{F}'_{i+1} \neq \emptyset$, we have to include each $X' \to E \in \mathcal{F}'_{i+1}$ to the new proof by deducing it from previously added items. This will guarantee the property $\mathcal{P}_{i+1}$ holds. For such a dependency of $\mathcal{F}'_{i+1}$, $E \in Z' \setminus X'$. If $E \in (Z' \setminus C) \cap Y' \subseteq X' \setminus Y'$, then $X' \to E \in \mathcal{F}'_{i_1}$ and since it was already added to the proof when f_{i_1} was considered, nothing is to be done. Otherwise, $E \in Z' \setminus X'Y' \subseteq Z' \setminus Y'$ and $Y' \to E$ was already added to the proof when f_{i_2} was considered. Let $X'' = X' \setminus Y'$, $Y'' = Y' \cap X'$ and $\{A_s \mid 1 \leq s \leq k\} = Y' \setminus X'$ as $|Y' \setminus X'| = k$ ($k = 0$ is allowed, it means no A_s's exist). With these notations, $Y' \to E = Y''A_1 \cdots A_k \to E$ and $\forall s \in [1..k] : X' \to A_s = X''Y'' \to A_s \in \mathcal{F}'_{i_1}$. Since elements of $\mathcal{F}'_{i_1}$ already occur in the constructed prefix of the proof, applying (U) with X'' as X and Y'' as Y is a valid step and the resulting dependency $X''Y'' \to E = Y' \to E$ can be added to the new proof.

□

Proof of Lemma 5

$V \to A_1 \cdots A_k$ can be deduced from $V \to A_1, \cdots, V \to A_k$ using (8) several times. We treat this as the first step for each case (with the corresponding V which can be XY, X or XZW).

We start with verifying (E1). Case $k = 0$ (and so $l = 0$) follows using the axiom and rule (3). Let $k > 0$, assume $XY \to A_1 \cdots A_k$, and $XY \twoheadrightarrow ZA_1 \cdots A_l$ for a $0 \leq l \leq k$. By using rule (1) we get $XY \to YA_1 \cdots A_k$, then $YA_1 \cdots A_k \twoheadrightarrow ZA_1 \cdots A_l$ follows according to rule (3). We finally get $YA_1 \cdots A_k \twoheadrightarrow Z$ by applying rule (5).

(E2) immediately follows from rule (6) (case $k = 0$ is irrelevant).

Now, for (E3), assume first that $k > 0$ and $XZW \to A_1 \cdots A_k$ and $XY \twoheadrightarrow ZA_1 \cdots A_k$ hold ($ZW \neq \emptyset$). We extend the first dependency using (1) to $XZW \to ZA_1 \cdots A_k$. Applying (7) then results $XY \twoheadrightarrow YZW$ and we get the desired constraint $XY \twoheadrightarrow ZW$ by rule (5). Case $k = 0$ corresponds to rule (4) which can be viewed as a special case of (7) as well ($Z \subseteq Y$).

Because the rules of system UE are sound, for each finite $\mathcal{F} \subset \mathbb{D}_c^+ \cup \mathbb{E}^+$, $\mathcal{F} \vdash_{UE} \neg(\mathrm{E}\square) \Longrightarrow$ $\mathcal{F} \vdash_{EA} \neg(\mathrm{E}\square)$. By using rule (6) of the extended Armstrong system repeatedly, starting from $\neg(\mathrm{E}\square)$, eg. $X \to A_1$ and $X \not\to A_1$ can be derived, resulting that $\mathcal{F}$ is contradictory.

□

Proof of Lemma 6

Let $\mathcal{F} \vdash_{EA} X \twoheadrightarrow Z$, $Z \setminus X \neq \emptyset$ and $\mathcal{F}$ is not contradictory. Note that $\mathcal{F}'$ is not contradictory as well, since it is equivalent to $\mathcal{F}$ in terms of the extended Armstrong system. We show that

for at least one element f'' of the set $\mathcal{H} = \{X \to U \mid U \subseteq Z \setminus X, U \neq \emptyset\}$, $\mathcal{F}' \vdash_{UE} f''$ holds by constructing a proof for it in system UE without using rule (E3) as $k = 0$. This verifies the second statement. Completeness then simply follows by using (E3) as $k = 0$ as one more step at the end of the proof, resulting $\mathcal{F}' \vdash_{UE} X \nrightarrow Z$.

Similarly to Lemma 4, we construct the new proof for an element of $\mathcal{H}$ in the system UE, parallel with the proof of $X \nrightarrow Z$ in the extended Armstrong system. Suppose the sequence $\langle f_1, \cdots, f_m \rangle$ is a proof of $X \nrightarrow Z$. Each item f_i can be either a functional dependency or an excluded constraint. For functional dependencies, the same method can be used as discussed in the proof of Lemma 4 so that elements of the set $\mathcal{F}'_i = \{V \to C \mid C \in W \setminus V\}$ already occur in the new proof before f_{i+1} is considered. For each excluded constraint $f_i = V \nrightarrow W$, $W \setminus V \neq \emptyset$ (since $\mathcal{F}$ is not contradictory[6]), therefore the set $\mathcal{F}''_i = \{V \to U \mid U \subseteq W \setminus V, U \neq \emptyset\}$ is nonempty. We ensure at least one element of $\mathcal{F}''_i$ occurs in the new proof (note that $\mathcal{F}''_i \neq \emptyset$) before f_{i+1} is considered. This property can be formalized by defining that $\mathcal{F}''_i = \emptyset$ for functional dependencies and $\mathcal{F}'_i = \emptyset$ for exlcuded constraints: for each i there exists a prefix $\langle f'_1, \cdots, f'_j \rangle$ in the new proof so that $\bigcup_{p=1}^{i} \mathcal{F}'_p \subseteq \bigcup_{q=1}^{j} \{f'_q\}$, $\bigcup_{p=1}^{i} (\mathcal{F}'_p \cup \mathcal{F}''_p) \supseteq \bigcup_{q=1}^{j} \{f'_q\}$ and $\forall p \in [1..i] : F''_p \neq \emptyset \Longrightarrow \mathcal{F}''_p \cap \bigcup_{q=1}^{j} \{f'_q\} \neq \emptyset$. Denote this property by $\mathcal{P}_i$. Since $\bigcup_{q=1}^{j} \{f'_q\} \subset \mathbb{D}_c^+ \cup \mathbb{E}^+$ holds, the proof to be constructed is valid in system UE over $\mathbb{D}_c^+ \cup \mathbb{E}^+$ if we use rules of system UE only. We start with the empty sequence as a prefix when no f_i's have been considered yet.

Let $0 < i \leq m$ and suppose by induction that $\langle f'_1, \cdots, f'_j \rangle$ is the prefix of the new proof already constructed according to $\langle f_1, \cdots, f_i \rangle$ with the property $\mathcal{P}_i$. Case $f_{i+1} \in \mathbb{D}$ (a positive element of $\mathcal{F}$, an instance of the axiom or a result of rule (1) or (2)) is already discussed in the proof of Lemma 4. If $f_{i+1} \in \mathbb{E}$, then $\mathcal{F}''_{i+1} \neq \emptyset$, we perform an (optional) extension resulting $\langle f'_1, \cdots, f'_j, f'_{j+1}, ..., f'_{j+h} \rangle$ so that $\mathcal{F}''_{i+1} \cap \bigcup_{q=1}^{j+h} \{f'_q\} \neq \emptyset$. This will complete the proof since $f_m = X \nrightarrow Z$, $\mathcal{F}''_m = \mathcal{H}$ and m is reached by induction.

Let $f_{i+1} \in \mathcal{F} \cap \mathbb{E}$, $f_i = X' \nrightarrow Z'$. In this case, $X' \nrightarrow Z' \setminus X' \in F''_i$ and we add this constraint to the new proof to make sure $\mathcal{P}_{i+1}$ holds.

If $f_{i+1} = Y' \nrightarrow Z' \, (Z' \setminus Y' \neq \emptyset)$ is derived by applying rule (3) to previous items $f_{i_1} = X' \to Y'$ and $f_{i_2} = X' \nrightarrow Z' \, (Z' \setminus X' \neq \emptyset)$. We have to extend the new proof with a $Y' \nrightarrow V'$ for a nonempty $V' \subseteq Z' \setminus Y'$. Let $X' \nrightarrow U' \in F''_{i_2} \, (U' \subseteq Z' \setminus X')$ that already occurs in the constructed prefix of the proof. Let $X'' = X' \setminus Y'$, $Y'' = X' \cap Y'$, $Z'' = U' \setminus Y' \subseteq Z' \setminus Y'$, $\{A_1, \cdots, A_k\} = Y' \setminus X'$ as $|Y' \setminus X'| = k$ ($k = 0$ is allowed resulting no $A'_s s$) and suppose $A_1, \cdots, A_l$ are the elements of $U' \cap Y'$, a subset of $Y' \setminus X'$. This way, $Z'' A_1 \cdots A_l = U'$ and $X'' Y'' \nrightarrow Z'' A_1 \cdots A_l = X' \nrightarrow U'$. For each $s \in [1..k]$, $X' \to A_s = X'' Y'' \to A_s \in \mathcal{F}'_{i_1}$ and therefore, they were added to the new proof when f_{i_1} was considered. Applying rule (E1) with X'' as X, Y'' as Y and Z'' as Z, we get $Y'' A_1 \cdots A_k \nrightarrow Z'' = Y' \nrightarrow Z''$ if $Z'' \neq \emptyset$. The proof being constructed is extended by this negated constraint, and $Z'' \subseteq Z' \setminus Y'$ ensures the desired property $\mathcal{P}_{i+1}$ holds. We show that Z'' cannot be empty. Suppose it is. Then, $X' \nrightarrow U' = X' \nrightarrow A_1 \cdots A_l$. Since each $X' \nrightarrow A_s$ was derived in system UE, we would get $\neg(\mathrm{E}\Box)$ which means F' is contradictory (see Lemma 5, the system UE is sound). But this contradicts our condition that F' is not contradictory, hence, $Z'' \neq \emptyset$.

If $f_{i+1} = X' \nrightarrow Y' Z' \, (Y' Z' \setminus X' \neq \emptyset)$ is a result of applying rule (4) to a previous constraint $f_{i'} = X' \nrightarrow Y'$, then by induction, a constraint $X' \nrightarrow U' \in F''_{i'} \, (U' \subseteq Y' \setminus X', U' \neq \emptyset)$ must already exist in the constructed prefix of the new proof. Since $U' \subseteq Y' \setminus X' \subseteq Y' Z' \setminus X'$, $F''_{i'} \subseteq F''_{i+1}$ and the property $\mathcal{P}_{i+1}$ holds without performing an extension.

[6] A self-contradictory constraint (a negated constraint of the form $XY \nrightarrow X$) can be deduced using the extended Armstrong implication system iff the inital set of constraints is contradictory. The 'only if' direction is trivial. To see the 'if' direction, apply rule (3) for the case $Y = Z$ or rule (6) for $Y = \emptyset$ or rule (7) for $X = Y$.

Consider an $f_{i+1} = X'Z' \nrightarrow Y' \, (Y' \setminus X'Z' \neq \emptyset)$ as the next case, derived from a previous constraint $f_{i'} = X'Y' \nrightarrow Y'Z'$ of the original proof using rule (5). By induction, a constraint $X'Z' \nrightarrow U' \in F''_{i'} \, (U' \subseteq Y'Z' \setminus X'Z', \, U' \neq \emptyset)$ must exist in the prefix already constructed. Again, no extension is neccessary for $\mathcal{P}_{i+1}$ to be true, since $Y'Z' \setminus X'Z' = Y' \setminus X'Z'$ and so $F''_{i'} = F''_{i+1}$.

Suppose now that $f_{i+1} = X' \nrightarrow Y' \, (Y' \setminus X' \neq \emptyset)$ and is derived applying rule (6) to previous items $f_{i_1} = X' \rightarrow Z'$ and $f_{i_2} = X' \; \not\rightarrow Y'Z'$. To ensure $\mathcal{P}_{i+1}$, we need to provide a constraint $X' \nrightarrow V' \in F''_{i+1} \, (V' \subseteq Y' \setminus X', \, V' \neq \emptyset)$ will be included into the new proof. We are supposing by induction that $X' \nrightarrow U' \in F''_{i_2}$ is already included for a nonempty set $U' \subseteq Y'Z' \setminus X'$ and we have nothing to do if $U' \subseteq Y' \setminus X'$. Let $\{A_1, \cdots A_k\} = (Z' \setminus X') \cap U'$ with the A_s's different ($k = 0$ is allowed), $Z'' = U' \setminus Z'$. Hence, $X' \nrightarrow U' = X' \nrightarrow Z''A_1 \cdots A_k$. Because of each $X' \rightarrow A_s \in F'_{i_1}$, they are already deduced and exist in the new proof. $Z'' \neq \emptyset$ can be shown similarly to the case of rule (3) above, therefore, rule (E2) can be applied with Z'' as Z and the result $X' \nrightarrow Z'' = X' \nrightarrow U' \setminus Z' \, (U' \setminus Z' \subseteq Y' \setminus X')$ can be added to the new proof ensuring $\mathcal{P}_{i+1}$.

The last case is when $f_{i+1} = X' \nrightarrow Y' \, (Y' \setminus X' \neq \emptyset)$ is a result of using rule (7) starting with previously derived constraints $f_{i_1} = Y' \rightarrow Z'$ and $f_{i_2} = X' \nrightarrow Z' \, (Z' \setminus X' \neq \emptyset)$. Providing a $X' \nrightarrow V' \in \mathcal{F}''_{i+1} \, (V' \subseteq Y' \setminus X', \, V' \neq \emptyset)$ exists in the new proof is neccessary for $\mathcal{P}_{i+1}$ to be true. By induction, an element $X' \nrightarrow U' \in F''_{i_2} \, (U' \subseteq Z' \setminus X', \, U' \neq \emptyset)$ is already deduced and included into the new proof being constructed. If $U' \setminus Y' = \emptyset$, then $U' \subseteq Y' \setminus X'$ and P_{i_2} implies P_{i+1}. Otherwise, let $\{A_1, \cdots A_k\} = U' \setminus Y'$ as $|U' \setminus Y'| = k \, (k > 0)$ and $Z'' = U' \cap Y'$, $Y'' = X' \cap Y'$, $X'' = X' \setminus Y'$, $W'' = Y' \setminus X'U'$. With these notations, $U' = Z''A_1 \cdots A_k$, $X' = X'' \setminus Y''$ and $Y' = Y''Z''W''$. For each $s \in [1..k]$, $Y' \rightarrow A_s = Y''Z''W'' \rightarrow A_s \in F'_{i_1}$ and they were added to the new proof when f_{i_1} was considered. Applying (E3) (with $k > 0$, X'' as X, Y'' as Y, Z'' as Z and W'' as W) to these together with $X' \nrightarrow U' = X''Y'' \nrightarrow Z''A_1 \cdots A_k$, the result is $X''Y'' \; \not\rightarrow Z''W''$. Note that $Y'' \cap Z'' = X' \cap Y' \cap U'$ which is empty since $U' \subseteq Z' \setminus X'$. Hence, $U' \cap Y' \subseteq Y' \setminus X'$ and $Z''W'' = (U' \cap Y') \cup (Y' \setminus X'U') = Y' \setminus X' \neq \emptyset$. Therefore, adding the deduced constraint to the new proof ensures $\mathcal{P}_{i+1}$ which completes our construction.

Construction of the proof for an element of $\mathcal{H}$ in the UE system can be performed by carrying over the above process for each element f_{i+1} of the original proof of $X \nrightarrow Z$ in the extended Armstrong system. Note we did not use the rule (E3) with $k = 0$ for any of the cases, verifying the second statement of the lemma.

Let us consider the final statement of this lemma regarding to the case when $\mathcal{F}$ is contradictory without containing trivial negated constraints. Suppose $\mathcal{F} \vdash_{EA} X \rightarrow Y$ for sets of attributes X and Y and $\mathcal{F} \vdash_{EA} X \nrightarrow Y$ at the same time. Let $\{C_1, \cdots, C_k\} = Y \setminus X$. Since rules (3)-(7) can not be used to deduce positive constraints, $\mathcal{F} \vdash_A X \rightarrow Y$ must hold which implies $\mathcal{F}' \vdash_U \{X \rightarrow C_i\}$ for each $i \in [1..k]$ according to Lemma 4. We show that either $\mathcal{F}' \vdash_{UE} X \nrightarrow C_1 \cdots C_l$ for an $l \in [1..k]$, or $\neg$(E$\square$) (with possibly other attributes) can be deduced instead. Let us start the construction of the proof of $X \nrightarrow C_1 \cdots C_k$ in system UE, paraller to the original proof of $X \nrightarrow Y$ the same way as discussed for the non-contradictory case. Reviewing the method, the only case of failure is when a self-contradictory constraint $X'Y' \nrightarrow X'$ arises in the original proof. If we let the empty set as being the right-hand side of negated constraints in UE for a while, the construction could be carried over, resulting that $X'Y' \nrightarrow \emptyset$ should be included to the new proof by using one of the rules (E1), (E2) or (E3). Instead of performing this step, it can easily be realized that the precondition for this step ($Z = \emptyset$ for (E1) or (E2) and $ZW = \emptyset$ for (E3)) is exactly $\neg$(E$\square$) with some attributes $A_1, \cdots, A_{k'}$. We conclude that $\neg$(E$\square$) must be already deduced if construction fails, otherwise, $X \nrightarrow C_1 \cdots C_k$ can be deduced for an $l \in [1..k]$ which corresponds to $\neg$(E$\square$) as well. Note we

still did not have to use (E3) with $k = 0$ for deducing $\neg(\mathrm{E}\square)$.

□

Proof of Lemma 7

Rules (S) and (T) are special cases of (U) with $k = 0$, $|X| = 1$ and $k = 1$, $|X| = 0$, respectively.

Let $k > 0$ and $|X| > 0$. We show that for each $0 \leq t \leq k$, $\{XY \rightarrow A_1, \cdots, XY \rightarrow A_k, YA_1 \cdots A_k \rightarrow B\} \vdash_{ST} XYA_1 \cdots A_t \rightarrow B$ hold ($t = 0$ corresponds our goal). Suppose, by induction for t that $XYA_1 \cdots A_{t+1} \rightarrow B$ is deducible. $XY \rightarrow A_{t+1}$ can be extended to get $XYA_1 \cdots A_t \rightarrow A_{t+1}$ using (S). Now we can immediately deduce ($XYA_1 \cdots A_t \rightarrow B$) by using (T). Case $t = 0$ follows by induction.

□

Proof of Lemma 8

Equivalence of {(U)} and {(S), (T)} was stated and proven by Lemma 7. We need to consider the rest of the rules dealing with negated dependenices. (NS) and (NT1) are special cases of (E1) with $|X| = 1$, $k = 0$ ($l = 0$) and $|X| = 0$, $k = 1$, $l = 0$, respectively. Similarly, (NT2) and ($*$) are special cases of (E3) with $k = 1$ and $k = 0$, respectiely.

(NT2S) can be easily simulated by extending $Y \rightarrow B$ to $YZ \rightarrow B$ using (S) and using (NT2) with $W = \emptyset$ to get $Y \nrightarrow Z$ afterwards.

Starting from $\neg(\mathrm{E}\square)$, $\neg(\square)$ can be derived by the repeated use of rule (NT2S). The reversed case is trivial since ($\square$) is a special case of ($\mathrm{E}\square$) with $k = 1$.

Consider (E1) with $k = 0$, $l = 0$, $|X| > 1$. This case can easily be simulated by repeatedly using (NS). Let $k > 0$. If $l > 0$, (NT2S) can be applied to remove A_i's ($1 \leq i \leq l$) step-by-step from the right-hand-side of $XY \nrightarrow ZA_1 \cdots A_l$ (dependency $XY \rightarrow A_i$ holds for each required i). The remaining case for (E1) is $k > 0$, $l = 0$. Applying (NT1) results $XYA_1 \nrightarrow Z$. We extend each dependency $XY \rightarrow A_j$ ($1 < j \leq k$) to $XYA_1 \cdots A_{j-1} \rightarrow A_j$ using (S) to allow (NT1) be applied repeatedly. This way we get $XYA_1 \cdots A_j \nrightarrow Z$ for each j. As $j = k$ reached, we get the desired result $YA_1 \cdots A_k \nrightarrow Z$ by using rule (NS) to remove X from the left-hand side.

In the case of (E2), only $k > 0$ is relevant and $X \nrightarrow ZA_1 \cdots A_j$ can be deduced for each $j \in [0..k]$ step-by-step using (NT2S). $j = 0$ corresponds to our goal $X \nrightarrow Z$.

Consider (E3) with $k > 0$. We extend $XZW \rightarrow A_k$ to $XZA_1 \cdots A_{k-1}W \rightarrow A_k$ using rule (S). Application of (N2) now results $XY \nrightarrow ZA_1 \cdots A_{k-1}W$ (attribute set $ZA_1 \cdots A_{k-1}$ corresponds to Z in the original formula of (N2)). Attributes $A_1 \cdots A_{k-1}$ of the right-hand side can be eliminated using (NT2S) repeatedly, resulting $XY \nrightarrow ZW$ as desired.

The remaining case for (E3) is $k = 0$ which can be simulated by the (sometimes repeated) use of ($*$). Note we did not need rule ($*$) for any other case. This verifies the last statement of the lemma, together with the fact that each rule (and the axiom ($\square$)) of NST corresponds to a special case of a rule (or axiom ($\mathrm{E}\square$)) of UE, and ($*$) is the only one for (E3) with $k = 0$.

□

Proof of Lemma 1

Trivial: Rules (P) and (Q) are special cases of (NS) and (NT1), respectively ($|Z| = 1$, $B = Z$). Rule (R) is a special case of rule (NT2) ($Z = \emptyset$, $|W| = 1$, $A = W$).

□

Proof of Lemma 2

Assume first that $\mathcal{F}$ is not contradictory (according to lemmas 6 and 8, this is equivalent with $\mathcal{F} \nvdash_{NST'} \neg(\square)$) and $\mathcal{F} \vdash_{NST'} X \twoheadrightarrow A$ holds. Consider a proof for the constraint $X \twoheadrightarrow A$. It clearly follows from the form of rules that the items of the proof can be reordered so that all functional dependencies precede the negated constraints. If we skip the items not needed for the deduction of $X \twoheadrightarrow A$, we notice that only the first negated constraint used in the proof is needed to be taken from $\mathcal{F}$ since there is no rule with two or more negated constraint as preconditions. Moreover, the rest of the negated constraints can be ordered so that each of them is derived applying a rule with the previous negated constraint (together with a positive constraint for some rules). Denote by $\langle \delta_1, \cdots, \delta_s, \varepsilon_t, \cdots, \varepsilon_1 \rangle$ this reduced, reordered proof using the NST' system ($\delta_i \in \mathbb{D}_c^+$, $\varepsilon_j \in \mathbb{D}_c^+$ for each $i \in [1..s]$ and $j \in [1..t]$). Note that $\varepsilon_1 = X \twoheadrightarrow A$ and since $\varepsilon_t \in \mathcal{F}$ and it is the only item ε_i with this property, $(\mathcal{F} \cap \mathbb{D}_c^+) \cup \{\varepsilon_t\} \vdash_{NST'} X \twoheadrightarrow A$ holds. We construct an equivalent proof using the system PQRST parallel to the original one. The positive part of the proof ($\langle \delta_1, \cdots, \delta_s \rangle$) is kept as the prefix since rules (S) and (T) are common for the two systems NST' and PQRST, the negative part ($\langle \varepsilon_t, \cdots, \varepsilon_1 \rangle$ can be replaced by another sequence $\langle \delta_{s+1}, \cdots \delta_r, \zeta_u, \cdots, \zeta_1 \rangle$ ($r \geq s$, $u \in \mathbb{N}^+$; $\delta_i \in \mathbb{D}_c^+$, $\zeta_j \in \mathbb{E}_c^+$ for each $i \in [s+1..r]$ and $j \in [1..u]$), resulting a valid proof using the PQRST system over $\mathbb{D}_c^+ \cup \mathbb{E}_c^+$. Construction is made by ensuring that for each $i \in [1..t]$ there exists a prefix $\langle \delta_1, \cdots, \delta_{h_i} \rangle$ and a postfix $\langle \zeta_{j_i}, \cdots, \zeta_1 \rangle$ of the new proof ($h_i \geq h_{i-1} \geq s$ and $j_i \geq j_{i-1}$ for each $i \in [2..t]$) so that $\langle \delta_1, \cdots, \delta_{h_i}, \zeta_{j_i}, \cdots, \zeta_1 \rangle$ is a valid proof for $(\mathcal{F} \cap \mathbb{D}_c^+) \cup \{\varepsilon_i\} \vdash_{PQRST} X \twoheadrightarrow A$. Furthermore, it is ensured that $\zeta_{j_i} = V \twoheadrightarrow A$ for each $\varepsilon_i = V \twoheadrightarrow W$ and $VA \rightarrow E \in \{\delta_1, \cdots, \delta_{h_i}\}$ for each $E \in |W \setminus A|$. Denote these properties by $\mathfrak{Q}_i$ for a particular $i \in [1..t]$. $\mathfrak{Q}_1$ holds by choosing ζ_1 as $\zeta_1 = \varepsilon_1$, $h_1 = s$ and $j_1 = 1$. t will be reached by induction, following the steps described below. If $\zeta_{j_t} = \varepsilon_t$, then $u = j_t$ and the proof is complete. Otherwise, $\zeta_{j_t} = Y \twoheadrightarrow A$, $\varepsilon_t = Y \twoheadrightarrow E\,(A \neq E)$ and according to $\mathfrak{Q}_t$, $YA \rightarrow E$ appears in the prefix already constructed. Adding $\zeta_{j_t+1} = \varepsilon_t$ as the first negated constraint to the postfix completes the proof (let $u = j_t + 1$) since ζ_{j_t} can be deduced applying rule (R) with $YA \rightarrow E$ and ε_t. This is the only possible application of rule (R) as it will be shown.

Assume by induction that a prefix $\langle \delta_1, \cdots, \delta_{h_i} \rangle$ and a postfix $\langle \zeta_{j_i}, \cdots, \zeta_1 \rangle$ of the new proof using the PQRST system is already constructed considering the postfix $\langle \varepsilon_i, \cdots, \varepsilon_1 \rangle$ of the original proof ($i \in [1..t-1]$), and $\mathfrak{Q}_i$ holds. We optionally perform an extension on the prefix or the postfix (or both), ensuring the property $\mathfrak{Q}_{i+1}$. Several subcases arise, depending on which rule was used to derive ε_i from ε_{i+1} ($\varepsilon_i \notin \mathcal{F}$ as seen above).

Assume $\varepsilon_i = Y \twoheadrightarrow Z$ and $\varepsilon_{i+1} = YC \twoheadrightarrow Z$ (rule (NS) was used in the original proof) and $\zeta_{j_i} = Y \twoheadrightarrow A$. Let $j_{i+1} = j_i + 1$, $\zeta_{j_{i+1}} = YC \twoheadrightarrow A$ (it is a valid step since ζ_{j_i} can be derived applying rule (P) with $\zeta_{j_{i+1}}$). To satisfy $\mathfrak{Q}_{i+1}$, we extend the prefix already constructed with each of the dependencies $YCA \rightarrow E\,(E \in Z \setminus A)$ not occuring in the prefix yet. Such a dependency can be deduced by applying rule (S) on $YA \rightarrow E$ which already exists in the prefix by induction. Note that $A \neq C$ since $\mathcal{F}$ is not contradictory ($A = C$ would mean $YC \rightarrow E$ for each $E \in Z \setminus C = Z$ and this would contradict $YC \twoheadrightarrow Z$).

The second case is when rule (NT1) was applied on $\varepsilon_{i+1} = Y \twoheadrightarrow Z$ and a $\delta_k = Y \rightarrow A'\,(k \leq s)$ to get $\varepsilon_i = YA' \twoheadrightarrow Z$. By induction, $\zeta_{j_i} = YA' \twoheadrightarrow A$ and $\forall E \in Z \setminus A\, \exists l \in [1..h_i] : \delta_l = YA'A \rightarrow E$. The postfix can be extended by $\zeta_{j_{i+1}} = \zeta_{j_i+1} = Y \twoheadrightarrow A$ since ζ_{j_i} can be deduced using rule (Q) in one step, starting with δ_k and this $\zeta_{j_{i+1}}$. Furthermore, $YA \rightarrow E$ can be deduced for each $E \in Z \setminus A$ using rule (S) and (T), starting with the already existing $Y \rightarrow A'$ and $YA'A \rightarrow E$. We extend the prefix with items $YA \rightarrow A'$ and $YA \rightarrow E$ if they do not occur in the prefix yet, ensuring $\mathfrak{Q}_{i+1}$.

The third and final case is of rule (NT2). Let $\varepsilon_i = Y \twoheadrightarrow ZW$, $\varepsilon_{i+1} = Y \twoheadrightarrow ZB$ and k such that $\delta_k = YZW \rightarrow B$ ($k \leq s$). By induction, $\zeta_{j_i} = Y \twoheadrightarrow A$ and $\forall E \in ZW \setminus A\, \exists l \in [1..h_i]$:

$\delta_l = YA \to E$. Let $j_{i+1} = j_i$ (no constraint is added to the postfix). Ensuring $YA \to E$ occurs in the prefix of the new proof for each $E \in ZB \setminus A$ provides Q_{i+1}. Since it occurs for each $E \in Z \setminus A$, only one dependency $YA \to B$ is needed to be included into the new proof if it does not occur yet (in this case, $A \neq B$). Rule (U) can be used to derive $YA \to B$ from δ_k and δ_l's mentioned above. According to Lemma 7, rule (U) can be simulated with rules (S) and (T). We extend the prefix of the new proof with steps of this deduction.

Construction of the new proof using system PQRST can be performed for a non-contradictory set by carrying over the above steps. If the initial set $\mathcal{F}$ is contradictory, then $\neg(\square)$ can be deduced using system NST'. For such a proof, the above transformation process may fail if a $\varepsilon_k = Y \nrightarrow Z$ exists such that $k > 1$ and $\mathcal{F} \vdash_{ST} Y \to E$ for each $E \in Z$ (see the case of rule (NS)). Selecting the first ε_k of the original proof (with the largest k possible) with this property and truncating the original proof at ε_k allows the transformation process to be carried out. We start by selecting an attribute $D \in Z$ arbitrarily for $\zeta_1 = Y \nrightarrow D$ and adding the steps of deduction of $Y \to E$ for each $E \in Z$ to the prefix $\langle \delta_1, \cdots, \delta_s \rangle$ of the new proof and go on as discussed. The process will not fail and both $Y \to D$ and $Y \nrightarrow D$ are deduced, indicating that $\mathcal{F}$ is contradictory.

□

Proof of Theorem 1

Soundness and completeness of ST follows using the soundness and completeness of the Armstrong system and lemmas 3, 4 and 7. Note that in Lemma 4, if $\mathcal{F} \in \mathbb{D}_c^+$ and $X \to Z \in \mathbb{D}_c^+$ as well, then $\mathcal{F} = \mathcal{F}'$ and there is exactly one $B \in Z \setminus X$ and $X \to Z = X \to B$ for this B. Therefore, we simply get $\mathcal{F} \vdash_{ST} \delta \iff \mathcal{F} \vdash_U \delta \iff \mathcal{F} \vdash_A \delta \iff \mathcal{F} \vDash \delta$.

□

Proof of Theorem 4

Part 1. Soundness and completeness of NST (without $(\square)$) follows using the soundness and completeness of the Armstrong system and lemmas 5, 6 and 8. Note that in Lemma 6, if $\mathcal{F} \subset \mathbb{D}_c^+ \cup \mathbb{E}^+$ and $X \nrightarrow Z \in \mathbb{E}^+$, then $\mathcal{F} = \mathcal{F}'$ and $X \nrightarrow Z = X \nrightarrow Z \setminus X$ with $Z \setminus X \neq \emptyset$. Similarly to Theorem 1, we get $\mathcal{F} \vdash_{NST} \delta \iff \mathcal{F} \vdash_{UE} \delta \iff \mathcal{F} \vdash_{EA} \delta \iff \mathcal{F} \vDash \delta$ for a non-contradictory set $\mathcal{F}$. Soundness holds regardless of $\mathcal{F}$ being contradictory or not, therefore, $\mathcal{F} \vdash_{UE} (\square) \Longrightarrow \mathcal{F} \vdash_{EA} (\square)$. The reverse direction for contradictory cases follows by Lemma 6, since $F = F' \subset \mathbb{D}_c^+ \cup \mathbb{E}^+$.

Part 2. Soundness over $\mathbb{D}_c^+ \cup \mathbb{E}^+$ and completeness over $\mathbb{D}_c^+$ is a trivial consequence of Part 1. Similarly to Part 1, $\mathcal{F}' = \mathcal{F}$. Using Lemmas 6 and 8 (focusing on their last statements), we get the following if $\mathcal{F}$ is not contradictory: $\mathcal{F} \Vdash X \nrightarrow Y \in \mathbb{E}^+ \Longrightarrow F \vdash_{UE} X \nrightarrow Y \Longrightarrow [\exists Z \subseteq Y,\ Z \neq \emptyset : \mathcal{F} \vdash_{UE} X \nrightarrow Z$ without using (E3) for $k = 0] \Longrightarrow [\mathcal{F} \vdash_{NST} X \nrightarrow Z$ for the same Z without using $(*)]$. For a contradictory set $\mathcal{F}$, we get $[\mathcal{F} \vdash_{UE} \neg(\mathrm{E}\square)$ without using (E3) for $k = 0] \Longrightarrow [\mathcal{F} \vdash_{NST} \neg(\square)$ without using $(*)]$.

□

Proof of Theorem 2

The statement immediately follows from Theorem 4 and Lemmas 1 and 2.

□

Proof of Theorem 3

Part 1. We show that applying (S) after (T) has been applied can be substituted by applying (T) twice after two subsequent application of (S). Suppose $YA \rightarrow B$ and $Y \rightarrow A$ hold. Applying (T) and then (S) results $Y \rightarrow B$ and $YC \rightarrow B$. Using (S) with both of the initial dependencies instead results $YCA \rightarrow B$ and $YC \rightarrow A$. By applying (T) with these two dependencies we get $YC \rightarrow B$. Hence, $Y \rightarrow B$ can be deduced now by applying (T) with the two initial dependencies.

Part 2. According to Part 1, all positive consequences can be deduced using (S) as many times as possible and then using (T) as many times as possible. It is clear that each application of (P), (Q) and (R) can be performed afterwards. We need to prove that an application of rule (R) after an application of rule (P) or (Q) can be substituted by another sequence so that (R) precedes all application of (P) and (Q).

As the first case, assume rule (R) is used directly after rule (Q). For three constraints $Y \rightarrow A$, $Y \nrightarrow B$ and $YAC \rightarrow B$, we get $YA \nrightarrow B$ using (Q) and then $YA \nrightarrow C$ using (R). For constructing an alternative way of deriving $YA \nrightarrow C$, note that $YC \rightarrow B$ must have been already deduced since it is a positive constraint that can be derived using rule (T). Applying rule (R) with $YC \rightarrow B$ and $Y \nrightarrow B$ first, we get $Y \nrightarrow C$ and the desired constraint $YA \nrightarrow C$ follows using rule (Q). A second application of (Q) the same way as in the original version results $YA \nrightarrow B$.

The second case is when rule (R) is used after rule (P): starting with constraints $YC \nrightarrow B$ and $YA \rightarrow B$, the constraint $Y \nrightarrow B$ follows using rule (P) and then $Y \nrightarrow A$ is derived using rule (R). Note that $YAC \rightarrow B$ must have been already deduced using rule (S). Starting with this constraint together with $YC \nrightarrow B$, constraint $YC \nrightarrow A$ follows using rule (R), and $Y \nrightarrow A$ can be derived using (P). A second application of (P) the same way as in the original version can be performed, resulting $Y \nrightarrow B$.

We conclude that (R) can precede any application of (P) and (Q) without any loss of deduction power. It is trivial that the order of (P) and (Q) is arbitrary.

If $|\mathcal{G}| = 1$ then the proof can be transformed the same way as in the proof of Lemma 2 since rules of system PQRST are special cases of those in system NST' (see Lemma 1). Rule (R) is used only once in the transformed proof.

□

Chapter 5

Graphical Reasoning

5.1 Graphical Rules and Sample Derivations

Recall the implication systems ST and PQRST with their rules (P), (Q), (R), (S) and (T) from section 4.1.

Since the ST implication system is sound and complete for non-trivial canonical functional dependencies, rules (S) and (T) can be used for deriving all implied functional dependencies given an inital set. Moreover, the PQRST implication system forms a sound and complete system for both positive and negative (excluded) non-trivial singleton functional constraints (see Theorem 2 in section 4.1)[1], rules (P), (Q) and (R) can be applied as complements of rules (S) and (T) when excluded functional constraints are taken into account.

These rules can be interpreted in terms of the graphical representation as well. A deduction step using one of them deals with a node of a higher-dimension object (eg. triangle as a two-dimensional object with one of its three verteces) and one or two of its borders (with one dimension lower, eg. edges of the same triagle as one-dimensional objects).

Graphical versions of rules are shown on Figure 1 for the triangular representation (case $Y = \{C\}$). The large grey arrows indicate the implication effect of each rule. Rule (S) is a simple extension rule and rule (T) can be called as "rotation rule" or "reduction rule". We may call the left-hand side of a functional dependency the *determinant* of it and the right-hand side the *determinate*. Rule (S) can be used to extend the determinant of a dependency resulting another dependency with one dimension higher, while rule (T) is used for *rotation*, that is, to replace the determinate of a functional dependency by the support of another functional dependency with one dimension higher (the small black arrow at B indicates support of $AC \rightarrow B$). Another possible way to interpret rule (T) is for reduction of the determinant of a higher-dimensional dependency by omitting an attribute if a dependency holds among the attributes of the determinant.

For excluded functional constraints, rule (Q) acts as the extension rule (needs support of a positive constraint, ie. 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 of rule (Q) (but without the need of support). Rule (Q) is also viewed as the negation of rule (S).

These graphical rules can be generalized to higher dimensional representations, where the number of attributes is more than 3. Figure 2 illustrates the rules (S) and (T) in terms of the

[1] Completeness hold when the initial set of constraints is not contradictory. However, the PQRST system can still be used to derive contradiction for each contradictory set according to the theorem. The same holds for system NST.

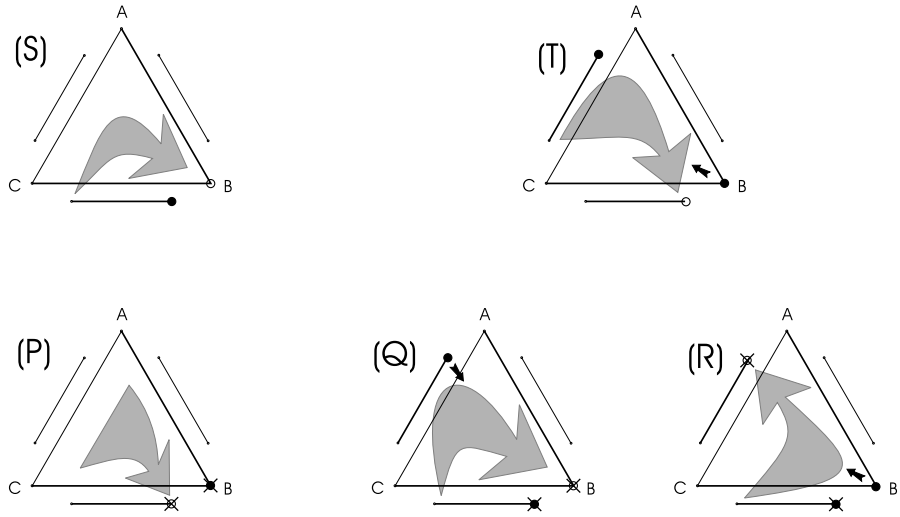

Fig. 1. Graphical versions of rules (P), (Q), (R), (S) and (T) in terms of the triangular representation. The small black arrows indicate support (neccessary context) while the large grey arrowns show the implication effects

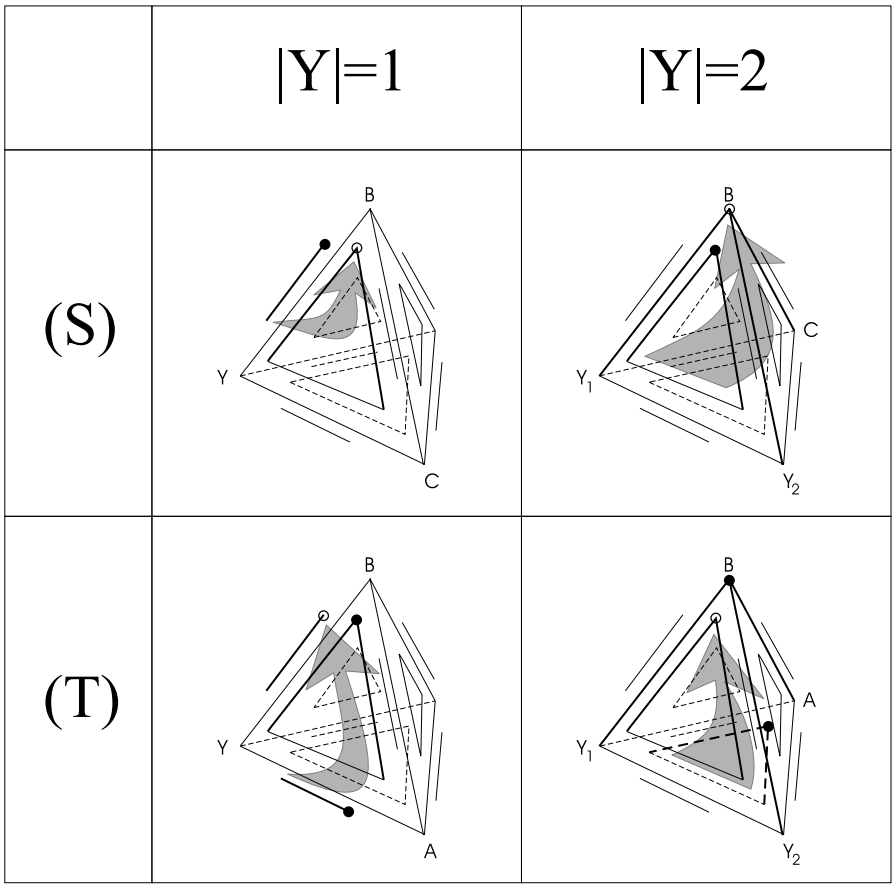

Fig. 2. Graphical rules (S) and (T) for different sizes of Y, presented in the tetrahedral representation

tetrahedral representation. Each of the rules has two versions, with $|Y| = 1$ and $|Y| = 2$. Rules for excluded constraints can be generalized the same way. This generalization can be carried on to larger schemata with raising the possible sizes of left-hand sides of constraints and so, extending figure 2 with additional columns of higher-dimensional operations.

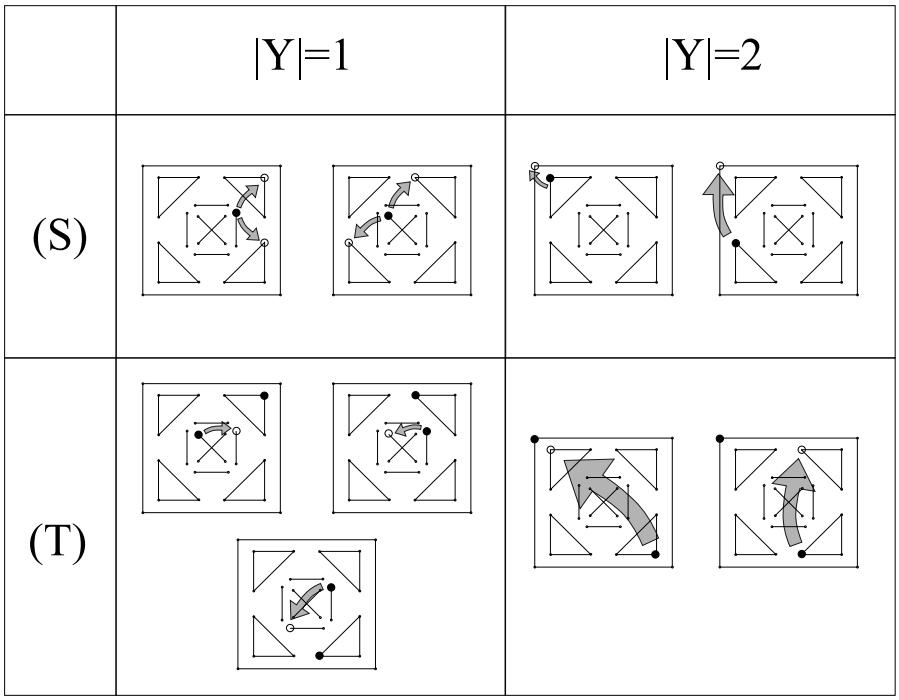

Fig. 3. Patterns of graphical rules (S) and (T) for the quadratic representation

However, graphical rules for higher dimensional spaces can hardly be handled using our human perception. The other way of constructing a representation for relational schemata with more attributes we proposed in Section 3 is to obtain a two-dimensional form by transforming the higher-dimensional representation for relational schemata with more attributes. Graphical rules for the triangular representation are discussed above. Although the rules for the redundant quadratic representation can be obtained quite straightforward, redundancy makes reasoning more difficult for the quadrary case. To avoid this difficulty, one may use the non-redundant quadratic representation, with rules adopted to it. Figure 3 shows the patterns of rules (S) and (T) for this case. The reason why two or three patterns arise for a single case is the broken symmetry of the quadratic representation compared to the tetrahedron (diagonals and sides of the sqare are conceptually equivalent edges).

To obtain patterns of graphical rules for quintary case, the pentagonal representation should be investigated. We are not considering all of the possible patterns, just present some examples on Figure 4. In general, for reasoning using the non-redundant two-dimensional representations for sets of functional constraints, the two key features to observe is which line is a line parallel to and which attribute a node "pointing towards". These features help to recognize which component is a border (edge, surface, etc.) of which one in the higher-dimensional version, and which components share a specific border. It can be easily discovered (recall Figure 2) that our graphical rules rely on these spatial relations.[2]

[2] For instance, a node of a border of an object can be extended to a node of the object itself using rule (S) or, by rule (T), a node of an object can be rotated into a node of another object sharing their borders on the

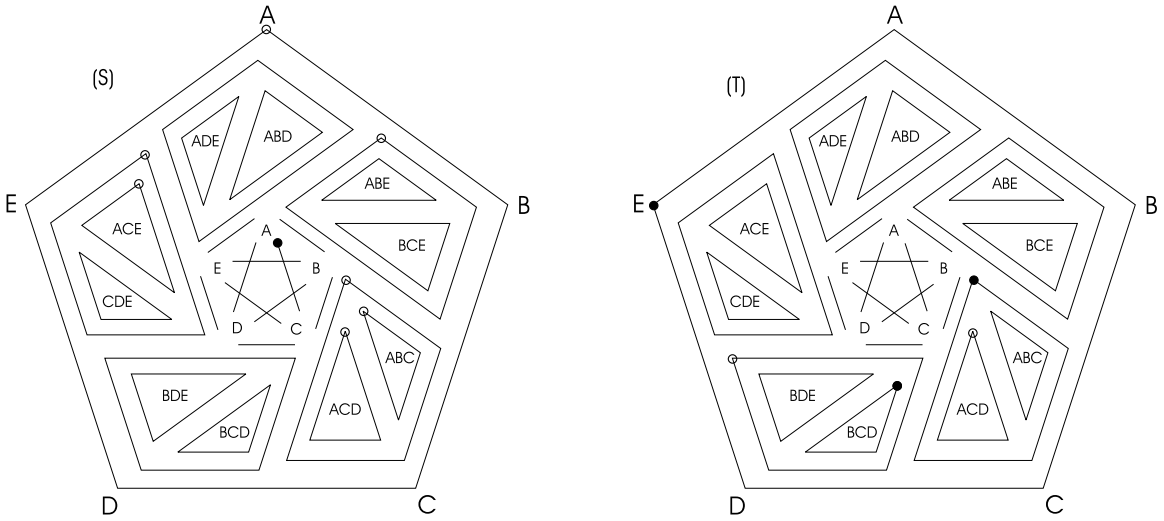

Fig. 4. Examples of repeated applications of graphical rules in the pentagonal representation. Left: rule (S) is used to get all implied functional dependencies of $C \to A$. Right: derivation $\{CD \to B,\ BCD \to A,\ ABCD \to E\} \vdash \{CD \to A,\ BCD \to E\}$ using rule (T) two times

For a sample derivation using the graphical rule (S) in the quadratic representations, refer to Figure 9 in Section 3. For getting the functional dependencies indicated by empty circles, rule (S) is applied four times.

Recall Figure 4 (also in Section 3) 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 (Q) 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 quadrary case, Figure 5 shows how transitivity can be simulated with these rules for the case $\{C \to B,\ C \to D,\ BD \to A\} \vdash C \to A$.

If the inital set contains non-singleton excluded constraints, we can use the NST' system instead of the PQRST (see section 4.3). However, these constraints are not included directly in the graphical representation since they do not correspond to the existence or abscence of a single functional dependency (they are treated as "meta-knowledge"). Instead, they can be interpreted as disjunctive conditions and can be stored separately until their right-hand sides are not reduced to one single attribute (some non-singleton negated constraints may not be reduced). Reduction of a right-hand side can be performed using rule (NT2S). Part 2 of Theorem 4 (Section 4) ensures that we do not loose information by replacing the original negated constraint with an implied one with its right-hand side reduced (ie. NST' is suitable instead of using the full NST including rule $(*)$). Another possible operation manipulating the right-hand side of an excluded dependency is applying rule (NT2) with $W \neq \emptyset$, resulting some extra knowledge about which functional dependencies may not hold.

Recall from Section 4.4 that system U with the single rule (U) forms a sound and complete implication system for the positive constraints, ie. functional dependencies. Instead of (S) and (T), (U) can also be directly applied for deduction using the graphical and spreadsheet representation. Considering different values for k and sizes of sets X and Y in rule (U), we get the 4 cases of rules (S) and (T) (presented on Figure 2) together with 6 additional cases (3 transitivity and one pseudotransitivity rules, a reduction and an extension rule) for four

side of the rest of their nodes (corresponding to their common determinants) if the appropriate node of the (one dimension higher) object they border is supporting it.

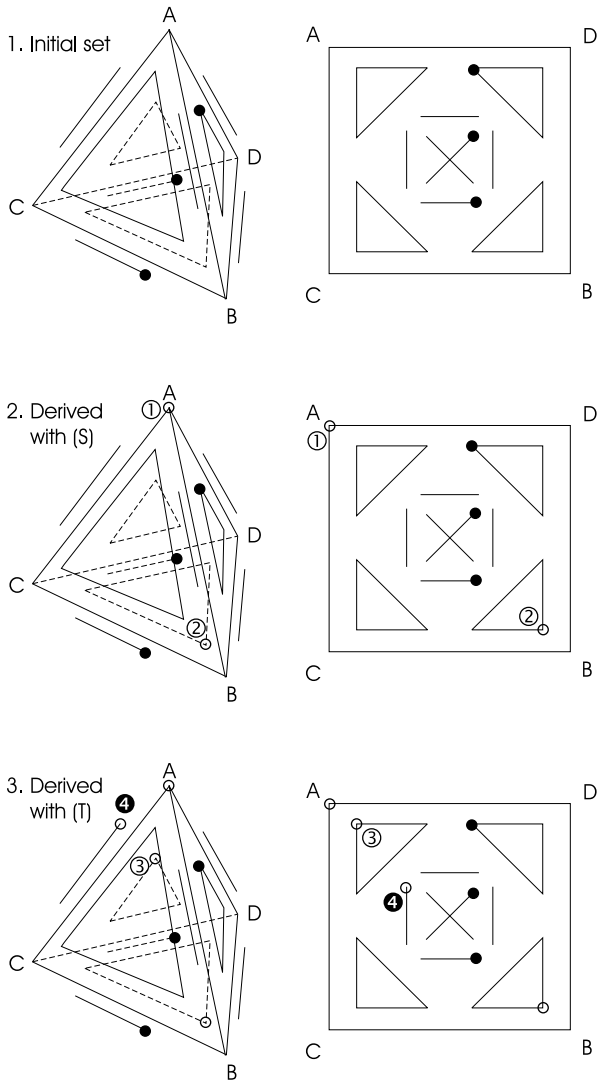

Fig. 5. Simulating transitivity with the ST implication system for the case $\{C \to B,\, C \to D,\, BD \to A\} \vdash C \to A$ (numbers show a possible order of deduction)

attributes. We have more possible patterns than using rules (S) and (T) but we may derive dependencies in less number of steps using rule (U). To include negated constraints, rules (P), (Q) and (R) can be used, or if non-singleton negated contstraints are allowed, rules (E1), (E2) and (E3) (UE implication system, see Section 4.4) can be used as the complements of (U) the same way as the rules for negated constraints of the NST' system.

When attributes are allowed to be declared as constants, graphical rules for zero-dimensional constraints are to be introduced. Implication systems ST and PQRST as well as NST', U and UE are capable to handle these type of constraints and the graphical representations can easily be extended (see Section 3.4.1). For instance, Figure 2 should be extended with a column for $|Y| = 0$, adding two more patterns to graphical rules (S) and (T). For the case of rule (U) the total number of patterns becomes 16 for four attributes.

5.2 Algorithms for Generating All Implied Functional Constraints

Simple Algorithms for the Triangular Representation Without considering the implication systems seriously, the triangular representation leads to rather simple acquisition algorithms:

Outside-Inside algorithm: We consider negated functional dependencies in the outer triangular.

1. If one of the dependencies $\{A, C\} \nrightarrow \{B\}$ is observed then we conclude directly $\{A\} \nrightarrow \{B\}$ and $\{C\} \nrightarrow \{B\}$.
2. Now we consider the remaining functional dependencies in the inner triangular (edges).
3. For disjoint X, Y, Z we may further directly conclude from $X \nrightarrow Z$ that either $Y \nrightarrow Z$ or $X \nrightarrow Y$ or both must be observable.

Inside-Outside algorithm: We consider the valid functional dependencies.

1. If the dependency $\{A\} \rightarrow \{B\}$ is valid then we conclude directly $\{A, C\} \rightarrow \{B\}$.
2. We conclude also the implied functional dependencies due to transitivity.
3. Now we consider the remaining functional dependencies in the outer triangular.

Mixed algorithm: The mixed algorithm applies the first steps of the inside-Outside and outside-inside algorithms in a mixed form.

The ST Algorithm for Sets of Functional Dependencies Considering the implication system ST (see Section 4.1), we give a modified and generalized version of the Inside-outside algorithm (just presented for the ternary case in terms of the triangular representation) for the use with also the tetrahedral, quadratic or other representations. This method is called *ST algorithm* and is already discussed as Part 1 of Theorem 3 in section 4.2:

1. Starting with the given initial set of non-trivial, canonical functional dependencies 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. Output the generated set.

Figure 2 (Section 5.1) can also be viewed as a table of elementary operations used by this algorithm in the quadrary case. As an example, Figure 6 shows how the complete set presented as quadrary case #64 (see section 2.3) can be derived from its given generating system using the ST algorithm.

Using rule (U) as many times as possible may be an alternative algorithm since the system U is sound and complete as well (see Section 4.4). This way we have more types of possible operations (10) and selecting the preconditions is usually more complex but we may derive dependencies in less number of steps (see also Section 5.1).

The STRPQ Algorithm for Sets of both Positive and Negative Constraints Now we incorporate excluded functional constraints into the system. Rules (P), (Q) and (R) can be applied as complements of rules (S) and (T), resulting the following algorithm called *STRPQ algorithm* (discussed as Part 2 of Theorem 3 in section 4.2):

1. Starting with the given initial set of non-trivial, singleton functional dependencies and excluded functional constraints input,

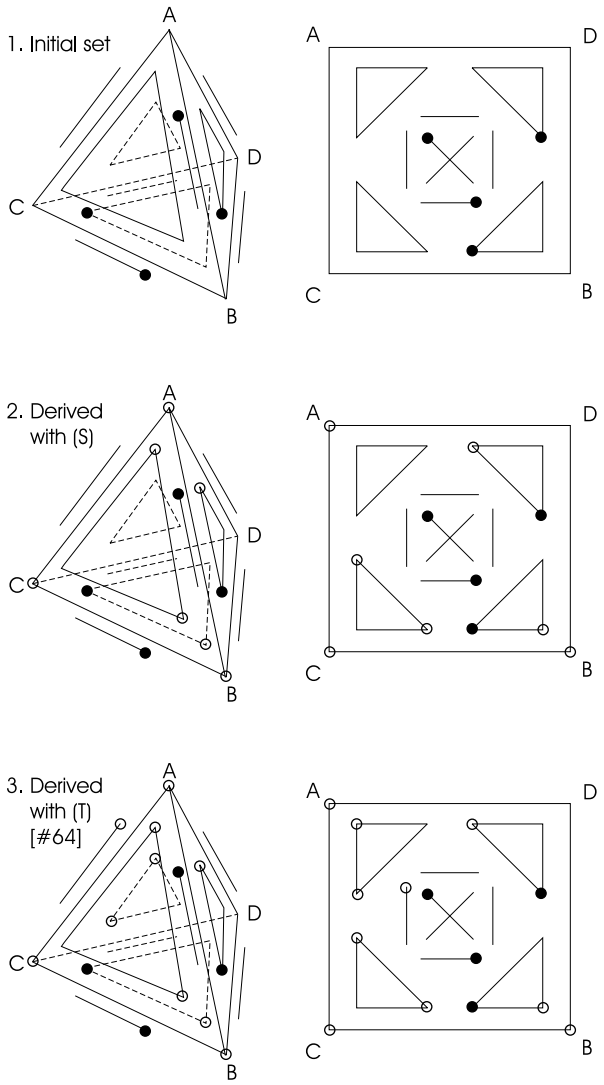

Fig. 6. The ST algorithm used for deriving dependencies of case #64 starting with its generating system

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 the excluded constraints using rules (P) and (Q) as many times as possible.
6. Output the generated set.

The algorithms just presented can be used for reasoning on sets of functional constraints, especially in terms of the graphical representations. The structure of the generalized triangular representations (triangular, tetrahedral, etc.) may also be used for designing a data structure representing sets of functional constraints for the algorithms.

Chapter 6

Extension to Spreadsheet Reasoning

Let us consider the spreadsheet representation for three attributes (see Section 2.3). 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 '.' to 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 1 shows how rules of the PQRST implication system (see Section 4.1) can be represented in the spreadsheet form.

BC → A	AC → B	AB → C	B → A	A → B	C → A	A → C	C → B	B → C	Implication impact of detected (negated) functional dependencies
.	$\vdash$ 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) $AB \to C,\ A \nrightarrow C \vdash A \nrightarrow B$

Table1. Example of the spreadsheet derivation of (negated) functional dependencies. Rules of the PQRST implication system in the spreadsheet form

The STRPQ algorithm presented in section 5.2 for the graphical representations provides a possible way for derivation of the full knowlegde a partial set holds in terms of the spreadsheet representation as well. Other implication systems can also be used.

The spreadsheet may be used for deriving contradictions as well. Contradictions occur whenever new constraints are introduced and the implication system allows to derive the opposite. We may indicate the contradiction by the symbol $\lightning$. The first case on Table 2 is an obvious one due to the rule system in the extended Armstrong system. The second one is due to rule (Q) of the PQRST system.

The elicitation algorithm presented in the introduction to this paper has got now the formal

BC → A	AC → B	AB → C	B → A	A → B	C → A	A → C	C → B	B → C	Implication impact of detected (negated) functional dependencies
.	.	.	.	1	.	0	.	$\vdash 0 \not\downarrow 1$	$\{A \rightarrow B, A \nrightarrow C\} \not\downarrow \{B \rightarrow C\}$
.		$\vdash 0 \not\downarrow 1$	.	1	.	0	.	.	$\{A \rightarrow B, A \nrightarrow C\} \not\downarrow \{AB \rightarrow C\}$

Table2. Deriving contradiction by spreadsheet reasoning

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 constraints which validity is still open. Using the approach of [ABDT98] 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.

Chapter 7

Conclusion

The problem whether there exist a simple and sophisticated representation of sets of constraints that supports reasoning on constraints is solved in this paper by introducing two more surveyable representations of constraint sets: graphical and spreadsheet representations. These representations require a different implication system than the classical Armstrong system. We, thus, introduced another system and could show (Theorem 1 and 2) its soundness and completeness.

This system has another really useful property: Constraint derivation may be ordered on the basis of sequences of rules. Derivation rule application can be described using the regular expression $(S)^*;(T)^*;(R)^*;((P)\|(Q))^*$. This order of rule application is extremely useful whenever we want to known whether the set of generated functional dependencies is full, i.e., consists of all dependencies that follow from the given initial system of functional dependencies. Based on this, we were able to generate all possible sets of initial functional dependencies for $n = 3, 4, 5$.

Graphical and spreadsheet reasoning support a simpler style of reasoning on constraint sets. Completeness and soundness of systems of functional dependencies and excluded functional dependencies becomes surveyable. Since database design approaches rely on completeness and soundness of constraint sets, our approach enables database designers to obtain better database design results.

Acknowledgements

We would like to thank Tibor Ásványi for his help in improving the efficiency of our PROLOG program, which generates and counts the sets of functional dependencies for small relation schemata and Zoltán Csaba Regéci for his assistance on running the program on a computer of MTA SZTAKI. We are also grateful to Andrea Molnár for her valuable comments on the illustration of the graphical rules and the tetrahedral representation.

Bibliography

[ABDT98] M. Albrecht, E. Buchholz, A. Düsterhöft, and 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*, pages 1–28. Springer, Berlin, 1998.

[AD93] P. Atzeni and V. De Antonellis. *Relational database theory.* Addison-Wesley, Redwood City, 1993.

[AT93] P. Atzeni and R. Torlone. A metamodel approach for the management of multiple models and the translation of schemes. *Information Systems*, 18(6):349–362, 1993.

[Cam02] R. Camps. From ternary relationship to relational tables: A case against common beliefs. *ACM SIGMOD Record, 31(2)*, pages 46–49, 2002.

[DK83] J. Demetrovics and G. O. H. Katona. Combinatorial problems of database models. In *Colloquia Mathematica Societatis Janos Bolyai 42, Algebra, Combinatorics and Logic in Computer Science*, pages 331–352, Györ, Hungary, 1983.

[DLM89] J. Demetrovics, L. O. Libkin, and I. B. Muchnik. Functional dependencies and the semilattice of closed classes. In *Proc. MFDBS'89, LNCS 364*, pages 136–147, 1989.

[Hal95] T. A. Halpin. *Conceptual schema and relational database design.* Prentice-Hall, Sydney, 1995.

[Kle98] M. Klettke. *Akquisition von Integritätsbedingungen in Datenbanken.* DISBIS 51. infix-Verlag, St. Augin, 1998.

[SYG96] V.C. Storey, H.L. Yang, and R.C. Goldstein. Semantic integrity constraints in knowledge-based database design systems. *Data & Knowledge Engineering*, 20:1–37, 1996.

[Tha87] B. Thalheim. Open problems in relational database theory. *Bull. EATCS*, 32:336 – 337, 1987.

[Tha00] B. Thalheim. *Entity-relationship modeling – Foundations of database technology.* Springer, Berlin, 2000. See also http://www.informatik.tu-cottbus.de/∼thalheim/HERM.htm.

[Tha02] B. Thalheim. Component construction of database schemes. In *Proc. ER'02, LNCS 2503*, pages 20–34, 2002.

[Yan86] C.-C. Yang. *Relational Databases.* Prentice-Hall, Englewood Cliffs, 1986.

Appendix

The PROLOG Program Generating and Calculating the Sets of FD's

case5.pl

```
%
% REMARKS:
% --------
%
%  Actual attribute names must be substituted for the variables A, B, C, ...
%  and must correspond to the order A @< B @< C @< ...
%
%  Dimension 100 represents infinity.
%
% Two types of representations are used for FD sets
%  (conversion is performed by case1fc, case2fc, ...):
%
%    1. Representation L: (lists)
%        multiple ordered lists corresponding to the components
%        of the graphical representation:
%           LA0, LB0, LC0, ..., LAB, LAC, LBC, ..., LABC, LABD, ..., ...
%              LA0, LB0, ...:          lists of zero-dimensional FD's (verteces)
%              LAB, LAC, LBC, ...:    lists of one-dimensional FD's (edges)
%              LABC, LACD, LBCD, ...: lists of two-dimensional FD's (triangles)
%              LABCD, LACDE, ...:     lists of three-dimensional FD's (tetrahedra)
%              ...
%
%    2. Representation FC: (flat, counted)
%        a single list consisting of a summary element (Crec)
%        and ordered sublists of the FD's grouped by dimensions:
%      [c(...,LenFL2,LenFL1,LenFL0), ..., FL2, FL1, FL0]
%              FL0: list of all zero-dimensional FD's, LenFL0 its length;
%              FL1: list of all one-dimensional FD's,  LenFL1 its length;
%              FL2: list of all two-dimensional FD's,  LenFL2 its length;
%              FL3: ...
%              ...
%
% for case1, case2, ...:
%
%  PA, PB, ... correspond to desired attribute dimensions
%  RA, RB, ... correspond to actual attribute dimensions
%

% generating or testing a member of a list

member(X,[X | _]).
member(X,[_ | Ys]) :- member(X,Ys).
```

```
% checking similarity of FD sets

similarcasef([],[],_,_).
similarcasef([L | Ls],[L2 | L2s],SC,DC) :- length(L,LN), length(L2,LN),
    replacel(L2,L2P,SC,DC), sort(L2P,L),
    similarcasef(Ls,L2s,SC,DC).

firstcasec([],[]).
firstcasec([[Crec|_] | _],Crec).

similarcasefcfound([[Crec|Ls1] | _],Crec,Ls,SC,DC) :-
    similarcasef(Ls1,Ls,SC,DC), !.
similarcasefcfound([[Crec|_] | Ss],Crec,Ls,SC,DC) :-
    similarcasefcfound(Ss,Crec,Ls,SC,DC).

% merging ordered lists

merge0([],L,L).
merge0([X|Xs],L,Ms) :- merge0_(L,X,Xs,Ms).

merge0_([],Y,Ys,[Y|Ys]).
merge0_([X|Xs],Y,Ys,Ms) :- X@<Y, !, Ms=[X|Ms2], merge0_(Xs,Y,Ys,Ms2).
merge0_([X|Xs],Y,Ys,[Y|Ms2]) :- merge0_(Ys,X,Xs,Ms2).

merge([],L,L,Len) :- length(L,Len).
merge([X|Xs],L,Ms,Len) :- merge_(L,X,Xs,Ms,Len).

merge_([],Y,Ys,[Y|Ys],Len) :- length([Y|Ys],Len).
merge_([X|Xs],Y,Ys,Ms,Len) :- X@<Y, !, Ms=[X|Ms2],
     merge_(Xs,Y,Ys,Ms2,Len1), Len is Len1+1.
merge_([X|Xs],Y,Ys,[Y|Ms2],Len) :- merge_(Ys,X,Xs,Ms2,Len1), Len is Len1+1.

merge([],Xs,Ys,Ms,Len) :- !, merge(Xs,Ys,Ms,Len).
merge(Xs,[],Ys,Ms,Len) :- !, merge(Xs,Ys,Ms,Len).
merge([X|Xs],[Y|Ys],Zs,Ms,Len) :- merge_(Zs,X,Xs,Y,Ys,Ms,Len).

merge_([],Y,Ys,Z,Zs,Ms,Len) :- merge_([Y|Ys],Z,Zs,Ms,Len).
merge_([X|Xs],Y,Ys,Z,Zs,Ms,Len) :- X@<Y, X@<Z, !, Ms=[X|Ms2],
     merge_(Xs,Y,Ys,Z,Zs,Ms2,Len1), Len is Len1+1.
merge_([X|Xs],Y,Ys,Z,Zs,Ms,Len) :- Y@<Z, !, Ms=[Y|Ms2],
     merge_(Ys,X,Xs,Z,Zs,Ms2,Len1), Len is Len1+1.
merge_([X|Xs],Y,Ys,Z,Zs,[Z|Ms2],Len) :- merge_(Zs,X,Xs,Y,Ys,Ms2,Len1),
     Len is Len1+1.

merge(Vs, Ws, Xs, Ys, Zs, LenZs) :-
    merge(Vs, Ws, Zs1, LenZs1), merge(Xs, Ys, Zs2, LenZs2), merge0(Zs1,Zs2,Zs),
    LenZs is LenZs1+LenZs2.
merge(Us, Vs, Ws, Xs, Ys, Zs, LenZs) :-
```

```
    merge(Us, Vs, Ws, Zs1, LenZs1), merge(Xs, Ys, Zs2, LenZs2),
    merge0(Zs1, Zs2, Zs), LenZs is LenZs1+LenZs2.
merge(Ts, Us, Vs, Ws, Xs, Ys, Zs, LenZs) :-
    merge(Ts, Us, Vs, Zs1, LenZs1), merge(Ws, Xs, Ys, Zs2, LenZs2),
    merge0(Zs1, Zs2, Zs), LenZs is LenZs1+LenZs2.
merge(Ss, Ts, Us, Vs, Ws, Xs, Ys, Zs, LenZs) :-
    merge(Ss, Ts, Us, Vs, Zs1, LenZs1), merge(Ws, Xs, Ys, Zs2, LenZs2),
    merge0(Zs1, Zs2, Zs), LenZs is LenZs1+LenZs2.
merge(Rs, Ss, Ts, Us, Vs, Ws, Xs, Ys, Zs, LenZs) :-
    merge(Rs, Ss, Ts, Us, Zs1, LenZs1), merge(Vs, Ws, Xs, Ys, Zs2, LenZs2),
    merge0(Zs1, Zs2, Zs), LenZs is LenZs1+LenZs2.
merge(Qs, Rs, Ss, Ts, Us, Vs, Ws, Xs, Ys, Zs, LenZs) :-
    merge(Qs, Rs, Ss, Ts, Us, Zs1, LenZs1), merge(Vs, Ws, Xs, Ys, Zs2, LenZs2),
    merge0(Zs1, Zs2, Zs), LenZs is LenZs1+LenZs2.
merge(Ps, Qs, Rs, Ss, Ts, Us, Vs, Ws, Xs, Ys, Zs, LenZs) :-
    merge(Ps, Qs, Rs, Ss, Ts, Zs1, LenZs1),
    merge(Us, Vs, Ws, Xs, Ys, Zs2, LenZs2),
    merge0(Zs1, Zs2, Zs), LenZs is LenZs1+LenZs2.
merge(Ks, Ls, Ms, Ns, Os, Ps, Qs, Rs, Ss, Ts, Us, Vs, Ws, Xs, Ys, Zs, LenZs) :-
    merge(Ks, Ls, Ms, Ns, Os, Ps, Qs, Rs, Zs1, LenZs1),
    merge(Ss, Ts, Us, Vs, Ws, Xs, Ys, Zs2, LenZs2),
    merge0(Zs1, Zs2, Zs), LenZs is LenZs1+LenZs2.
merge(Fs, Gs, Hs, Is, Js, Ks, Ls, Ms, Ns, Os, Ps, Qs, Rs, Ss, Ts, Us,
    Vs, Ws, Xs, Ys, Zs, LenZs) :-
    merge(Fs, Gs, Hs, Is, Js, Ks, Ls, Ms, Ns, Os, Zs1, LenZs1),
    merge(Ps, Qs, Rs, Ss, Ts, Us, Vs, Ws, Xs, Ys, Zs2, LenZs2),
    merge0(Zs1, Zs2, Zs), LenZs is LenZs1+LenZs2.

% generating or testing permutations of attributes

perm([X, Y],X,Y, RX,RY, RX,RY).
perm([Y, X],X,Y, RX,RY, RY,RX).

perm([X | YZ],X,Y,Z, RX,RY,RZ, RX,R1,R2) :- perm(YZ,Y,Z, RY,RZ, R1,R2).
perm([Y | XZ],X,Y,Z, RX,RY,RZ, RY,R1,R2) :- perm(XZ,X,Z, RX,RZ, R1,R2).
perm([Z | XY],X,Y,Z, RX,RY,RZ, RZ,R1,R2) :- perm(XY,X,Y, RX,RY, R1,R2).

perm([W | XYZ],W,X,Y,Z, RW,RX,RY,RZ, RW,R1,R2,R3) :-
    perm(XYZ,X,Y,Z, RX,RY,RZ, R1,R2,R3).
perm([X | WYZ],W,X,Y,Z, RW,RX,RY,RZ, RX,R1,R2,R3) :-
    perm(WYZ,W,Y,Z, RW,RY,RZ, R1,R2,R3).
perm([Y | WXZ],W,X,Y,Z, RW,RX,RY,RZ, RY,R1,R2,R3) :-
    perm(WXZ,W,X,Z, RW,RX,RZ, R1,R2,R3).
perm([Z | WXY],W,X,Y,Z, RW,RX,RY,RZ, RZ,R1,R2,R3) :-
    perm(WXY,W,X,Y, RW,RX,RY, R1,R2,R3).

perm([V | WXYZ],V,W,X,Y,Z, RV,RW,RX,RY,RZ, RV,R1,R2,R3,R4) :-
    perm(WXYZ,W,X,Y,Z, RW,RX,RY,RZ, R1,R2,R3,R4).
```

```
perm([W | VXYZ],V,W,X,Y,Z, RV,RW,RX,RY,RZ, RW,R1,R2,R3,R4) :-
    perm(VXYZ,V,X,Y,Z, RV,RX,RY,RZ, R1,R2,R3,R4).
perm([X | VWYZ],V,W,X,Y,Z, RV,RW,RX,RY,RZ, RX,R1,R2,R3,R4) :-
    perm(VWYZ,V,W,Y,Z, RV,RW,RY,RZ, R1,R2,R3,R4).
perm([Y | VWXZ],V,W,X,Y,Z, RV,RW,RX,RY,RZ, RY,R1,R2,R3,R4) :-
    perm(VWXZ,V,W,X,Z, RV,RW,RX,RZ, R1,R2,R3,R4).
perm([Z | VWXY],V,W,X,Y,Z, RV,RW,RX,RY,RZ, RZ,R1,R2,R3,R4) :-
    perm(VWXY,V,W,X,Y, RV,RW,RX,RY, R1,R2,R3,R4).

perm([U | VWXYZ],U,V,W,X,Y,Z, RU,RV,RW,RX,RY,RZ, RU,R1,R2,R3,R4,R5) :-
    perm(VWXYZ,V,W,X,Y,Z, RV,RW,RX,RY,RZ, R1,R2,R3,R4,R5).
perm([V | UWXYZ],U,V,W,X,Y,Z, RU,RV,RW,RX,RY,RZ, RV,R1,R2,R3,R4,R5) :-
    perm(UWXYZ,U,W,X,Y,Z, RU,RW,RX,RY,RZ, R1,R2,R3,R4,R5).
perm([W | UVXYZ],U,V,W,X,Y,Z, RU,RV,RW,RX,RY,RZ, RW,R1,R2,R3,R4,R5) :-
    perm(UVXYZ,U,V,X,Y,Z, RU,RV,RX,RY,RZ, R1,R2,R3,R4,R5).
perm([X | UVWYZ],U,V,W,X,Y,Z, RU,RV,RW,RX,RY,RZ, RX,R1,R2,R3,R4,R5) :-
    perm(UVWYZ,U,V,W,Y,Z, RU,RV,RW,RY,RZ, R1,R2,R3,R4,R5).
perm([Y | UVWXZ],U,V,W,X,Y,Z, RU,RV,RW,RX,RY,RZ, RY,R1,R2,R3,R4,R5) :-
    perm(UVWXZ,U,V,W,X,Z, RU,RV,RW,RX,RZ, R1,R2,R3,R4,R5).
perm([Z | UVWXY],U,V,W,X,Y,Z, RU,RV,RW,RX,RY,RZ, RZ,R1,R2,R3,R4,R5) :-
    perm(UVWXY,U,V,W,X,Y, RU,RV,RW,RX,RY, R1,R2,R3,R4,R5).

% replacing attribute letters of an FD set

replacel([],[],_,_).
replacel([d(US,AS) | Ss],[d(UD,AD) | Ds],SC,DC) :-
    repl_attrlist(US,UD1,SC,DC), sort(UD1,UD), repl_atom(AS,AD,SC,DC),
    replacel(Ss,Ds,SC,DC).

repl_attrlist([],[],_,_).
repl_attrlist([S | Ss],[D | Ds],SC,DC) :-
    repl_atom(S,D,SC,DC),
    repl_attrlist(Ss,Ds,SC,DC).

repl_atom(S,S,[],[]) :- !.
repl_atom(S,D,[S | _],[D | _]) :- !.
repl_atom(S,D,[_ | SCs],[_ | DCs]) :- !, repl_atom(S,D,SCs,DCs).

% -------------------
% Generating the sets
% -------------------

% checking the validity of rule (T) [adding YA->B won't violate?]

tvalid(Y,A,B,LYA,LYB) :-
    (member(d(Y,A),LYA) -> member(d(Y,B),LYB); true).

% unary cases
```

```
case1([],_A,100,PA) :- PA>0.
case1([d([],A)],A,0,0).

case1fc([c(LenFL0),FL0],A,RA) :-
    case1(FL0,A,RA,RA),
    length(FL0,LenFL0).

cases1fc(Ss,N,A,RA) :-
    findall(S,case1fc(S,A,RA),Ss1),
    length(Ss1,N),sort(Ss1,Ss).

reducecases1fc([],[],_,_).
reducecases1fc([[Crec|Ls] | Ss],RSs,X,RX) :-
    firstcasec(Ss,Crec),
    similarcasefcfound(Ss,Crec,Ls,[X],[X]),
    !, reducecases1fc(Ss,RSs,X,RX).
reducecases1fc([S | Ss],[S | RSs],X,RX) :-
    reducecases1fc(Ss,RSs,X,RX).

% binary cases

case2(LAB,LA0,LB0,A,B,RA,RB,PA,PB) :-
    case1(LA0,A,RAm,PA),
    case1(LB0,B,RBm,PB),

    (RBm<100 -> (tvalid([],A,B,LA0,LB0), LAB=[d([A],B) | LAB1], RB=RBm)
     ; ((PB=<1, tvalid([],A,B,LA0,LB0), LAB=[d([A],B) | LAB1], RB=1);
        (LAB=LAB1, RB=100))),
    (RAm<100 -> (tvalid([],B,A,LB0,LA0), LAB1=[d([B],A)], RA=RAm)
     ; ((PA=<1, tvalid([],B,A,LB0,LA0), LAB1=[d([B],A)], RA=1);
        (LAB1=[], RA=100))).


case2fc([c(LenFL1,LenFL0),FL1,FL0],A,B,RA,RB) :-
    case2(FL1,LA0,LB0,A,B,RA,RB,RA,RB),
    length(FL1,LenFL1), merge(LA0,LB0,FL0,LenFL0).

cases2fc(Ss,N,A,B,RA,RB) :-
    findall(S,case2fc(S,A,B,RA,RB),Ss1),
    length(Ss1,N),sort(Ss1,Ss).

reducecases2fc([],[],_,_,_,_).
reducecases2fc([[Crec|Ls] | Ss],RSs,X,Y,RX,RY) :-
    firstcasec(Ss,Crec),
    perm(PermXY,X,Y,RX,RY,RX,RY),
    similarcasefcfound(Ss,Crec,Ls,[X,Y],PermXY),
    !, reducecases2fc(Ss,RSs,X,Y,RX,RY).
```

```
reducecases2fc([S | Ss],[S | RSs],X,Y,RX,RY) :-
    reducecases2fc(Ss,RSs,X,Y,RX,RY).

% ternary cases

case3(LABC,LAB,LAC,LBC,LA0,LB0,LC0,A,B,C,RA,RB,RC,PA,PB,PC) :-

    case2(LAB,LA0,LB0,A,B,RA1,RB1,PA,PB),
    case2(LAC,LA0,LC0,A,C,RA2,RC1,PA,PC),
    case2(LBC,LB0,LC0,B,C,RB2,RC2,PB,PC),

    RAm is min(RA1,RA2), RBm is min(RB1,RB2), RCm is min(RC1,RC2),

    (RCm<100 -> (tvalid([A],B,C,LAB,LAC), tvalid([B],A,C,LAB,LBC),
                 LABC=[d([A,B],C) | LABC1], RC=RCm)
     ; ((PC=<2, tvalid([A],B,C,LAB,LAC), tvalid([B],A,C,LAB,LBC),
                 LABC=[d([A,B],C) | LABC1], RC=2); (LABC=LABC1, RC=100))),
    (RBm<100 -> (tvalid([A],C,B,LAC,LAB), tvalid([C],A,B,LAC,LBC),
                 LABC1=[d([A,C],B) | LABC2], RB=RBm)
     ; ((PB=<2, tvalid([A],C,B,LAC,LAB), tvalid([C],A,B,LAC,LBC),
                 LABC1=[d([A,C],B) | LABC2], RB=2); (LABC1=LABC2, RB=100))),
    (RAm<100 -> (tvalid([B],C,A,LBC,LAB), tvalid([C],B,A,LBC,LAC),
                 LABC2=[d([B,C],A)], RA=RAm)
     ; ((PA=<2, tvalid([B],C,A,LBC,LAB), tvalid([C],B,A,LBC,LAC),
                 LABC2=[d([B,C],A)], RA=2); (LABC2=[], RA=100))).

case3fc([c(LenFL2,LenFL1,LenFL0),FL2,FL1,FL0],A,B,C,RA,RB,RC) :-
    case3(FL2,LAB,LAC,LBC,LA0,LB0,LC0,A,B,C,RA,RB,RC,RA,RB,RC),
    length(FL2,LenFL2), merge(LAB,LAC,LBC,FL1,LenFL1),
    merge(LA0,LB0,LC0,FL0,LenFL0).

cases3fc(Ss,N,A,B,C,RA,RB,RC) :-
    findall(S,case3fc(S,A,B,C,RA,RB,RC),Ss1),
    length(Ss1,N),sort(Ss1,Ss).

reducecases3fc([],[],_,_,_,_,_,_).
reducecases3fc([[Crec|Ls] | Ss],RSs,X,Y,Z,RX,RY,RZ) :-
    firstcasec(Ss,Crec),
    perm(PermXYZ,X,Y,Z,RX,RY,RZ,RX,RY,RZ),
    similarcasefcfound(Ss,Crec,Ls,[X,Y,Z],PermXYZ),
    !, reducecases3fc(Ss,RSs,X,Y,Z,RX,RY,RZ).
reducecases3fc([S | Ss],[S | RSs],X,Y,Z,RX,RY,RZ) :-
    reducecases3fc(Ss,RSs,X,Y,Z,RX,RY,RZ).

% quadrary cases

case4(LABCD,LABC,LABD,LACD,LBCD,LAB,LAC,LAD,LBC,LBD,LCD,
      LA0,LB0,LC0,LD0,
```

```
      A,B,C,D,RA,RB,RC,RD,PA,PB,PC,PD) :-
    case3(LABC,LAB,LAC,LBC,LA0,LB0,LC0,A,B,C,RA1,RB1,RC1,PA,PB,PC),
    case3(LABD,LAB,LAD,LBD,LA0,LB0,LD0,A,B,D,RA2,RB2,RD1,PA,PB,PD),
    case3(LACD,LAC,LAD,LCD,LA0,LC0,LD0,A,C,D,RA3,RC2,RD2,PA,PC,PD),
    case3(LBCD,LBC,LBD,LCD,LB0,LC0,LD0,B,C,D,RB3,RC3,RD3,PB,PC,PD),

    RAm is min(RA1,min(RA2,RA3)), RBm is min(RB1,min(RB2,RB3)),
    RCm is min(RC1,min(RC2,RC3)), RDm is min(RD1,min(RD2,RD3)),

    (RDm<100 -> (tvalid([A,B],C,D,LABC,LABD),
                 tvalid([A,C],B,D,LABC,LACD),
         tvalid([B,C],A,D,LABC,LBCD),
         LABCD=[d([A,B,C],D) | LABCD1], RD=RDm)
     ; ((PD=<3, tvalid([A,B],C,D,LABC,LABD),
        tvalid([A,C],B,D,LABC,LACD),
        tvalid([B,C],A,D,LABC,LBCD),
        LABCD=[d([A,B,C],D) | LABCD1], RD=3);
    (LABCD=LABCD1, RD=100))),
    (RCm<100 -> (tvalid([A,B],D,C,LABD,LABC),
         tvalid([A,D],B,C,LABD,LACD),
         tvalid([B,D],A,C,LABD,LBCD),
         LABCD1=[d([A,B,D],C) | LABCD2], RC=RCm)
     ; ((PC=<3, tvalid([A,B],D,C,LABD,LABC),
        tvalid([A,D],B,C,LABD,LACD),
        tvalid([B,D],A,C,LABD,LBCD),
        LABCD1=[d([A,B,D],C) | LABCD2], RC=3);
    (LABCD1=LABCD2, RC=100))),
    (RBm<100 -> (tvalid([A,C],D,B,LACD,LABC),
         tvalid([A,D],C,B,LACD,LABD),
         tvalid([C,D],A,B,LACD,LBCD),
         LABCD2=[d([A,C,D],B) | LABCD3], RB=RBm)
     ; ((PB=<3, tvalid([A,C],D,B,LACD,LABC),
        tvalid([A,D],C,B,LACD,LABD),
        tvalid([C,D],A,B,LACD,LBCD),
        LABCD2=[d([A,C,D],B) | LABCD3], RB=3);
    (LABCD2=LABCD3, RB=100))),
    (RAm<100 -> (tvalid([B,C],D,A,LBCD,LABC),
         tvalid([B,D],C,A,LBCD,LABD),
         tvalid([C,D],B,A,LBCD,LACD),
         LABCD3=[d([B,C,D],A)], RA=RAm)
     ; ((PA=<3, tvalid([B,C],D,A,LBCD,LABC),
        tvalid([B,D],C,A,LBCD,LABD),
        tvalid([C,D],B,A,LBCD,LACD),
        LABCD3=[d([B,C,D],A)], RA=3);
    (LABCD3=[], RA=100))).

case4fc([c(LenFL3,LenFL2,LenFL1,LenFL0),FL3,FL2,FL1,FL0],
        A,B,C,D,RA,RB,RC,RD) :-
```

```
    case4(FL3,LABC,LABD,LACD,LBCD,LAB,LAC,LAD,LBC,LBD,LCD,
      LA0,LB0,LC0,LD0,
      A,B,C,D,RA,RB,RC,RD,RA,RB,RC,RD),
    length(FL3,LenFL3), merge(LABC,LABD,LACD,LBCD,FL2,LenFL2),
    merge(LAB,LAC,LAD,LBC,LBD,LCD,FL1,LenFL1),
    merge(LA0,LB0,LC0,LD0,FL0,LenFL0).

cases4fc(Ss,N,A,B,C,D,RA,RB,RC,RD) :-
    findall(S,case4fc(S,A,B,C,D,RA,RB,RC,RD),Ss1),
    length(Ss1,N),sort(Ss1,Ss).

reducecases4fc([],[],_,_,_,_,_,_,_,_).
reducecases4fc([[Crec|Ls] | Ss],RSs,W,X,Y,Z,RW,RX,RY,RZ) :-
    firstcasec(Ss,Crec),
    perm(PermWXYZ,W,X,Y,Z,RW,RX,RY,RZ,RW,RX,RY,RZ),
    similarcasefcfound(Ss,Crec,Ls,[W,X,Y,Z],PermWXYZ),
    !, reducecases4fc(Ss,RSs,W,X,Y,Z,RW,RX,RY,RZ).
reducecases4fc([S | Ss],[S | RSs],W,X,Y,Z,RW,RX,RY,RZ) :-
    reducecases4fc(Ss,RSs,W,X,Y,Z,RW,RX,RY,RZ).

% quintary cases

case5(LABCDE,LABCD,LABCE,LABDE,LACDE,LBCDE,
      LABC,LABD,LABE,LACD,LACE,LADE,LBCD,LBCE,LBDE,LCDE,
      LAB,LAC,LAD,LAE,LBC,LBD,LBE,LCD,LCE,LDE,
      LA0,LB0,LC0,LD0,LE0,
      A,B,C,D,E,RA,RB,RC,RD,RE,PA,PB,PC,PD,PE) :-
    case4(LABCD,LABC,LABD,LACD,LBCD,LAB,LAC,LAD,LBC,LBD,LCD,
      LA0,LB0,LC0,LD0,
      A,B,C,D,RA1,RB1,RC1,RD1,PA,PB,PC,PD),
    case4(LABCE,LABC,LABE,LACE,LBCE,LAB,LAC,LAE,LBC,LBE,LCE,
      LA0,LB0,LC0,LE0,
      A,B,C,E,RA2,RB2,RC2,RE1,PA,PB,PC,PE),
    case4(LABDE,LABD,LABE,LADE,LBDE,LAB,LAD,LAE,LBD,LBE,LDE,
      LA0,LB0,LD0,LE0,
      A,B,D,E,RA3,RB3,RD2,RE2,PA,PB,PD,PE),
    case4(LACDE,LACD,LACE,LADE,LCDE,LAC,LAD,LAE,LCD,LCE,LDE,
      LA0,LC0,LD0,LE0,
      A,C,D,E,RA4,RC3,RD3,RE3,PA,PC,PD,PE),
    case4(LBCDE,LBCD,LBCE,LBDE,LCDE,LBC,LBD,LBE,LCD,LCE,LDE,
      LB0,LC0,LD0,LE0,
      B,C,D,E,RB4,RC4,RD4,RE4,PB,PC,PD,PE),

    RAm is min(RA1,min(RA2,min(RA3,RA4))),
    RBm is min(RB1,min(RB2,min(RB3,RB4))),
    RCm is min(RC1,min(RC2,min(RC3,RC4))),
    RDm is min(RD1,min(RD2,min(RD3,RD4))),
    REm is min(RE1,min(RE2,min(RE3,RE4))),
```

```
(REm<100 -> (tvalid([A,B,C],D,E,LABCD,LABCE),
             tvalid([A,B,D],C,E,LABCD,LABDE),
     tvalid([A,C,D],B,E,LABCD,LACDE),
     tvalid([B,C,D],A,E,LABCD,LBCDE),
     LABCDE=[d([A,B,C,D],E) | LABCDE1], RE=REm)
 ; ((PE=<4, tvalid([A,B,C],D,E,LABCD,LABCE),
        tvalid([A,B,D],C,E,LABCD,LABDE),
        tvalid([A,C,D],B,E,LABCD,LACDE),
        tvalid([B,C,D],A,E,LABCD,LBCDE),
        LABCDE=[d([A,B,C,D],E) | LABCDE1], RE=4);
    (LABCDE=LABCDE1, RE=100))),
(RDm<100 -> (tvalid([A,B,C],E,D,LABCE,LABCD),
         tvalid([A,B,E],C,D,LABCE,LABDE),
     tvalid([A,C,E],B,D,LABCE,LACDE),
     tvalid([B,C,E],A,D,LABCE,LBCDE),
     LABCDE1=[d([A,B,C,E],D) | LABCDE2], RD=RDm)
 ; ((PD=<4, tvalid([A,B,C],E,D,LABCE,LABCD),
        tvalid([A,B,E],C,D,LABCE,LABDE),
        tvalid([A,C,E],B,D,LABCE,LACDE),
    tvalid([B,C,E],A,D,LABCE,LBCDE),
        LABCDE1=[d([A,B,C,E],D) | LABCDE2], RD=4);
    (LABCDE1=LABCDE2, RD=100))),
(RCm<100 -> (tvalid([A,B,D],E,C,LABDE,LABCD),
     tvalid([A,B,E],D,C,LABDE,LABCE),
     tvalid([A,D,E],B,C,LABDE,LACDE),
     tvalid([B,D,E],A,C,LABDE,LBCDE),
     LABCDE2=[d([A,B,D,E],C) | LABCDE3], RC=RCm)
 ; ((PC=<4, tvalid([A,B,D],E,C,LABDE,LABCD),
    tvalid([A,B,E],D,C,LABDE,LABCE),
        tvalid([A,D,E],B,C,LABDE,LACDE),
    tvalid([B,D,E],A,C,LABDE,LBCDE),
        LABCDE2=[d([A,B,D,E],C) | LABCDE3], RC=4);
    (LABCDE2=LABCDE3, RC=100))),
(RBm<100 -> (tvalid([A,C,D],E,B,LACDE,LABCD),
     tvalid([A,C,E],D,B,LACDE,LABCE),
     tvalid([A,D,E],C,B,LACDE,LABDE),
     tvalid([C,D,E],A,B,LACDE,LBCDE),
     LABCDE3=[d([A,C,D,E],B) | LABCDE4], RB=RBm)
 ; ((PB=<4, tvalid([A,C,D],E,B,LACDE,LABCD),
    tvalid([A,C,E],D,B,LACDE,LABCE),
        tvalid([A,D,E],C,B,LACDE,LABDE),
    tvalid([C,D,E],A,B,LACDE,LBCDE),
        LABCDE3=[d([A,C,D,E],B) | LABCDE4], RB=4);
    (LABCDE3=LABCDE4, RB=100))),
(RAm<100 -> (tvalid([B,C,D],E,A,LBCDE,LABCD),
     tvalid([B,C,E],D,A,LBCDE,LABCE),
     tvalid([B,D,E],C,A,LBCDE,LABDE),
```

```
         tvalid([C,D,E],B,A,LBCDE,LACDE),
         LABCDE4=[d([B,C,D,E],A)], RA=RAm)
     ; ((PA=<4, tvalid([B,C,D],E,A,LBCDE,LABCD),
        tvalid([B,C,E],D,A,LBCDE,LABCE),
            tvalid([B,D,E],C,A,LBCDE,LABDE),
        tvalid([C,D,E],B,A,LBCDE,LACDE),
            LABCDE4=[d([B,C,D,E],A)], RA=4);
        (LABCDE4=[], RA=100))).

case5fc([c(LenFL4,LenFL3,LenFL2,LenFL1,LenFL0),FL4,FL3,FL2,FL1,FL0],
    A,B,C,D,E,RA,RB,RC,RD,RE) :-
    case5(FL4,LABCD,LABCE,LABDE,LACDE,LBCDE,
      LABC,LABD,LABE,LACD,LACE,LADE,LBCD,LBCE,LBDE,LCDE,
      LAB,LAC,LAD,LAE,LBC,LBD,LBE,LCD,LCE,LDE,
      LA0,LB0,LC0,LD0,LE0,
      A,B,C,D,E,RA,RB,RC,RD,RE,RA,RB,RC,RD,RE),
    length(FL4,LenFL4),
    merge(LABCD,LABCE,LABDE,LACDE,LBCDE,FL3,LenFL3),
    merge(LABC,LABD,LABE,LACD,LACE,LADE,LBCD,LBCE,LBDE,LCDE,FL2,LenFL2),
    merge(LAB,LAC,LAD,LAE,LBC,LBD,LBE,LCD,LCE,LDE,FL1,LenFL1),
    merge(LA0,LB0,LC0,LD0,LE0,FL0,LenFL0).

cases5fc(Ss,N,A,B,C,D,E,RA,RB,RC,RD,RE) :-
    findall(S,case5fc(S,A,B,C,D,E,RA,RB,RC,RD,RE),Ss1),
    length(Ss1,N),sort(Ss1,Ss).

reducecases5fc([],[],_,_,_,_,_,_,_,_,_,_).
reducecases5fc([[Crec|Ls] | Ss],RSs,V,W,X,Y,Z,RV,RW,RX,RY,RZ) :-
    firstcasec(Ss,Crec),
    perm(PermVWXYZ,V,W,X,Y,Z,RV,RW,RX,RY,RZ,RV,RW,RX,RY,RZ),
    similarcasefcfound(Ss,Crec,Ls,[V,W,X,Y,Z],PermVWXYZ),
    !, reducecases5fc(Ss,RSs,V,W,X,Y,Z,RV,RW,RX,RY,RZ).
reducecases5fc([S | Ss],[S | RSs],V,W,X,Y,Z,RV,RW,RX,RY,RZ) :-
    reducecases5fc(Ss,RSs,V,W,X,Y,Z,RV,RW,RX,RY,RZ).
```

case5present.pl

```
%
% REMARKS:
% --------
%
%  Actual attribute names must be substituted for the variables A, B, C, ...
%  and must correspond to the order A @< B @< C @< ...
%  For generating the FD sets, call 'presentallclasses(a,b,...).' where
%  each letter corresponds to an attribute [(a,b,c) for the ternary case,
%  (a,b,c,d) for the quadrary and so on].
%
%  Uncommenting 'dim1(0).' includes zero-dimensional constraints (allows
%  the attributes to be constant)
```

```
%
%  Dimension number 100 represents infinity.
%
%  Cases are generated grouped by dimensions of attributes.
%  The FC representation is used for sets of FD's (see case5.pl)
%
%  By default, the tabular representation is used for output.
%  Uncommenting 'writecase' and the relating lines will output the sets
%  in their conventional format.
%
%  All cases are generated first (with equivalent sets treated as distinct)
%  and their list is reduced afterwards ('Different cases').
%
%  A class is generated only if the attribute dimensions correspond to the
%  ordering RA =< RB =< RC ... This is enough for generating the different
%  cases (where cases equivalent up to permutation are treated as one).
%  However, to get the number of all cases, the number of cases of a
%  class is multiplied by a coefficient for the number of total classes
%  with the same number of dimensions (eg.: for the ternary class
%  [A]=1, [B]=1, [C]=2 the total number of cases is multiplied by
%  3 since the two other classes [A]=2, [B]=1, [C]=1 and [A]=1, [B]=2, [C]=1
%  are not generated). Certainly, this is not applied to the number of
%  different cases (the reduced list).
%

% Write a set of FD's

writecasesfc([],_Vars,N,N) :- !, write('No cases.'), nl.
writecasesfc(Ss,Vars,N,N1) :- writecasesfc_(Ss,Vars,N,N1).

writecasesfc_([],_Vars, N, N).
writecasesfc_([[_Crec|S]|Ss],Vars, N, N1) :-
    write('#'), write(N), write(':  '),
    writecasefbinary(S,Vars),
%    write(' { '), writecase(S), write('} '),
    nl,
    N_ is N+1,
    writecasesfc_(Ss,Vars,N_,N1).

%writecase([]).
%writecase([L|Ls]) :- writedeplist(L), writecase(Ls).
%
%writedeplist([]).
%writedeplist([D|Ds]) :- writedep(D), write(' '), writedeplist(Ds).
%
%writedep(d(L,A)) :- writeattrlist(L), write('->'), write(A).
%
%writeattrlist([]).
```

```
%writeattrlist([A|As]) :- write(A), writeattrlist(As).

% Binary (spreadsheet) representation

% n=5:

writecasefbinary([FL4,FL3,FL2,FL1,FL0], [A, B, C, D, E]) :-
    writebinary(FL4, A, B, C, D, E), write(' '),

    writebinary(FL3, A, B, C, D), write(' '),
    writebinary(FL3, A, B, C, E), write(' '),
    writebinary(FL3, A, B, D, E), write(' '),
    writebinary(FL3, A, C, D, E), write(' '),
    writebinary(FL3, B, C, D, E), write(' '),

    writebinary(FL2, A, B, C), write(' '),
    writebinary(FL2, A, B, D), write(' '),
    writebinary(FL2, A, B, E), write(' '),
    writebinary(FL2, A, C, D), write(' '),
    writebinary(FL2, A, C, E), write(' '),
    writebinary(FL2, A, D, E), write(' '),
    writebinary(FL2, B, C, D), write(' '),
    writebinary(FL2, B, C, E), write(' '),
    writebinary(FL2, B, D, E), write(' '),
    writebinary(FL2, C, D, E), write(' '),

    writebinary(FL1, A, B), write(' '),
    writebinary(FL1, A, C), write(' '),
    writebinary(FL1, A, D), write(' '),
    writebinary(FL1, A, E), write(' '),
    writebinary(FL1, B, C), write(' '),
    writebinary(FL1, B, D), write(' '),
    writebinary(FL1, B, E), write(' '),
    writebinary(FL1, C, D), write(' '),
    writebinary(FL1, C, E), write(' '),
    writebinary(FL1, D, E), write(' '),

    writebinary(FL0, A), write(' '),
    writebinary(FL0, B), write(' '),
    writebinary(FL0, C), write(' '),
    writebinary(FL0, D), write(' '),
    writebinary(FL0, E).

% n=4:

writecasefbinary([FL3,FL2,FL1,FL0], [A, B, C, D]) :-
    writebinary(FL3, A, B, C, D), write(' '),
```

```
    writebinary(FL2, A, B, C), write(' '),
    writebinary(FL2, A, B, D), write(' '),
    writebinary(FL2, A, C, D), write(' '),
    writebinary(FL2, B, C, D), write(' '),

    writebinary(FL1, A, B), write(' '),
    writebinary(FL1, A, C), write(' '),
    writebinary(FL1, A, D), write(' '),
    writebinary(FL1, B, C), write(' '),
    writebinary(FL1, B, D), write(' '),
    writebinary(FL1, C, D), write(' '),

    writebinary(FL0, A), write(' '),
    writebinary(FL0, B), write(' '),
    writebinary(FL0, C), write(' '),
    writebinary(FL0, D).

% n=3:

writecasefbinary([FL2,FL1,FL0], [A, B, C]) :-
    writebinary(FL2, A, B, C), write(' '),

    writebinary(FL1, A, B), write(' '),
    writebinary(FL1, A, C), write(' '),
    writebinary(FL1, B, C), write(' '),

    writebinary(FL0, A), write(' '),
    writebinary(FL0, B), write(' '),
    writebinary(FL0, C).

% n=2:

writecasefbinary([FL1,FL0], [A, B]) :-
    writebinary(FL1, A, B), write(' '),

    writebinary(FL0, A), write(' '),
    writebinary(FL0, B).

% n=1:

writecasefbinary([FL0], [A]) :-
    writebinary(FL0, A).

% auxilliary:

writebinary(L, A, B, C, D, E) :-
    (member(d([B, C, D, E], A), L) -> write(1); write(0)),
    (member(d([A, C, D, E], B), L) -> write(1); write(0)),
```

```
    (member(d([A, B, D, E], C), L) -> write(1); write(0)),
    (member(d([A, B, C, E], D), L) -> write(1); write(0)),
    (member(d([A, B, C, D], E), L) -> write(1); write(0)).

writebinary(L, A, B, C, D) :-
    (member(d([B, C, D], A), L) -> write(1); write(0)),
    (member(d([A, C, D], B), L) -> write(1); write(0)),
    (member(d([A, B, D], C), L) -> write(1); write(0)),
    (member(d([A, B, C], D), L) -> write(1); write(0)).

writebinary(L, A, B, C) :-
    (member(d([B, C], A), L) -> write(1); write(0)),
    (member(d([A, C], B), L) -> write(1); write(0)),
    (member(d([A, B], C), L) -> write(1); write(0)).

writebinary(L, A, B) :-
    (member(d([B], A), L) -> write(1); write(0)),
    (member(d([A], B), L) -> write(1); write(0)).

writebinary(L, A) :-
    (member(d([], A), L) -> write(1); write(0)).



% Possible dimensions for the unary, binary, ternary, quadrary and
%    quintary cases

dim1(100).
%dim1(0).             % UNCOMMENT THIS LINE TO ALLOW ATTRIBUTES TO BE CONSTANT
dim2(RA) :- dim1(RA).
dim2(1).
dim3(RA) :- dim2(RA).
dim3(2).
dim4(RA) :- dim3(RA).
dim4(3).
dim5(RA) :- dim4(RA).
dim5(4).

% Dimension combinations [ (*) RA =< RB =< RC =< RD =< RE ]

dimcomb1([RA]) :- dim1(RA).
dimcomb2([RA, RB]) :- dim2(RA), dim2(RB), RA=<RB.
dimcomb3([RA, RB, RC]) :- dim3(RA), dim3(RB), RA=<RB, dim3(RC), RB=<RC.
dimcomb4([RA, RB, RC, RD]) :- dim4(RA), dim4(RB), RA=<RB, dim4(RC),
                              RB=<RC, dim4(RD), RC=<RD.
dimcomb5([RA, RB, RC, RD, RE]) :- dim5(RA), dim5(RB), RA=<RB, dim5(RC),
                                  RB=<RC, dim5(RD), RC=<RD, dim5(RE), RD=<RE.
```

```
% Nr. of all cases must be multiplied by the number of possible
%     permutations of the attributes to compensate for (*)

permnr(L,N) :- permnr_(L,Fprod,Pprod,_,_), N is Fprod//Pprod.
permnr_([_I],1,1,2,2).
permnr_([I,I|Is],Fprod,Pprod,Fnext,Pnext) :- !,
    permnr_([I|Is],Fp,Pp,Fn,Pn), Fprod is Fp*Fn, Pprod is Pp*Pn,
    Fnext is Fn+1, Pnext is Pn+1.
permnr_([_I,J|Js],Fprod,Pprod,Fnext,Pnext) :- !,
    permnr_([J|Js],Fp,Pp,Fn,_Pn), Fprod is Fp*Fn, Pprod is Pp,
    Fnext is Fn+1, Pnext=2.

% -----------------
% MAIN ENTRY POINTS
% -----------------

presentallclasses(A) :-
    write('n=1'), nl, nl,
    findall(R,dimcomb1(R),Rs), presentclasses(Rs, [A], 0, 0, N, RN),
    writesummary(N,RN).

presentallclasses(A, B) :-
    A@<B,
    write('n=2'), nl, nl,
    findall(R,dimcomb2(R),Rs), presentclasses(Rs, [A, B], 0, 0, N, RN),
    writesummary(N,RN).

presentallclasses(A, B, C) :-
    A@<B, B@<C,
    write('n=3'), nl, nl,
    findall(R,dimcomb3(R),Rs), presentclasses(Rs, [A, B, C], 0, 0, N, RN),
    writesummary(N,RN).

presentallclasses(A, B, C, D) :-
    A@<B, B@<C, C@<D,
    write('n=4'), nl, nl,
    findall(R,dimcomb4(R),Rs), presentclasses(Rs, [A, B, C, D], 0, 0, N, RN),
    writesummary(N,RN).

presentallclasses(A, B, C, D, E) :-
    A@<B, B@<C, C@<D, D@<E,
    write('n=5'), nl, nl,
    findall(R,dimcomb5(R),Rs), presentclasses(Rs, [A, B, C, D, E], 0, 0, N, RN),
    writesummary(N,RN).

% Class presentation

writeheaderitem(A, RA) :- write('['), write(A), write(']='), write(RA).
```

```
writeclassheader([A],[RA]) :- !, writeheaderitem(A, RA), nl.
writeclassheader([A|As],[RA|RAs]) :- writeheaderitem(A, RA),
                                     write(', '), writeclassheader(As,RAs).


presentclasses([], _Vars, N, RN, N, RN).
presentclasses([C|Cs], Vars, InN, InRN, OutN, OutRN) :-
    writeclassheader(Vars, C),
    write('---------------------------------'), nl,
    presentclass(C, Vars, InN, InRN, N1, RN1),
    write('---------------------------------'), nl,
    nl,
    presentclasses(Cs, Vars, N1, RN1, OutN, OutRN).

presentclass(Class, Vars, InN, InRN, OutN, OutRN) :-
    write('All cases: '), nl,
    casesfc(Ss_,N,Vars,Class),
    writecasesfc(Ss_,Vars,InN,_), permnr(Class,PN), OutN is InN+N*PN,
    write(PN), write(' * '), write(N), write(' case(s).'), nl, nl,
    write('Different cases: '), nl,
    reducecasesfc(Ss_,Ss,Vars,Class),
    writecasesfc(Ss,Vars,InRN,OutRN),
    length(Ss,RN), write(RN), write(' case(s).'), nl, nl.

% Summary

writesummary(N,RN) :-
    write('---------------------------------'), nl,
    write('---------------------------------'), nl,
    write('TOTAL CASES: '), write(N),
    write('; TOTAL DIFFERENT CASES: '), write(RN), nl, nl.

% Translate parameter passing

casesfc(Ss,N,[A],[RA]) :-
    cases1fc(Ss,N,A,RA).

casesfc(Ss,N,[A,B],[RA,RB]) :-
    cases2fc(Ss,N,A,B,RA,RB).

casesfc(Ss,N,[A,B,C],[RA,RB,RC]) :-
    cases3fc(Ss,N,A,B,C,RA,RB,RC).

casesfc(Ss,N,[A,B,C,D],[RA,RB,RC,RD]) :-
    cases4fc(Ss,N,A,B,C,D,RA,RB,RC,RD).

casesfc(Ss,N,[A,B,C,D,E],[RA,RB,RC,RD,RE]) :-
```

```
    cases5fc(Ss,N,A,B,C,D,E,RA,RB,RC,RD,RE).


reducecasesfc(Ss,RSs,[A],[RA]) :-
    reducecases1fc(Ss,RSs,A,RA).

reducecasesfc(Ss,RSs,[A,B],[RA,RB]) :-
    reducecases2fc(Ss,RSs,A,B,RA,RB).

reducecasesfc(Ss,RSs,[A,B,C],[RA,RB,RC]) :-
    reducecases3fc(Ss,RSs,A,B,C,RA,RB,RC).

reducecasesfc(Ss,RSs,[A,B,C,D],[RA,RB,RC,RD]) :-
    reducecases4fc(Ss,RSs,A,B,C,D,RA,RB,RC,RD).

reducecasesfc(Ss,RSs,[A,B,C,D,E],[RA,RB,RC,RD,RE]) :-
    reducecases5fc(Ss,RSs,A,B,C,D,E,RA,RB,RC,RD,RE).
```