Using Concept Lattices to mine functional dependencies*

J. Baixeries
Departament de LSI
Universitat Politècnica de Catalunya
jbaixer@lsi.upc.es

May 19, 2003

Abstract

Concept Lattices have been proved to be a valuable tool to represent the knowlegde in a database. In this paper we show how functional dependencies in databases can be extracted using Concept Lattices, not preprocessing the original database, but providing a new closure operator. We also prove that this method generalizes the previous methods and closure operators that are being used to find association rules in binary databases.

1 Introduction

Concept Lattices are a useful tool to represent logical implications in datasets, to find frequent itemsets, and in general, to analyze the underground knowledge that lies behind large amounts of data. Concept lattices are formed by concepts that, broadly speaking, are groups of objects classified together having common attributes. Apart from seeming to be close to the human representation of knowledge, the importance of this model relies on the fact that it is supported by a consistent mathematical theory, which is originally based on Birkhoff's lattice theory [3], later developed by Ganter and Wille [11], as well as providing a comprehensive graphical interface, that can show structural relations in a given set of data just at a glance. This model has been successfully applied to the "market basket analysis", which, in data mining terminology, means the finding of sets that appear together in a large database [2]. The goal is to provide information on when the buying of an item determines that another item is likely to be bought too.

Currently the main issues concerning concept lattices have been focused on finding optimal algorithmic solutions in order to compute closures [12]. Some other authors have been dealing with the incremental update of the concept lattice when the set of tuples is dynamically incremented, in order to find an optimal method to create the concept lattice in an incremental way, as well as to provide a conceptual framework [19], [16], [17]. Relating concept lattices to other theoretical frameworks has been proposed in [1], in which an equivalence between rules formed in a concept lattice out of a set of binary tuples, and Horn clauses formed out of a set of models (which is in fact the set of binary tuples seen as a set of models) has been proved.

On the other hand, functional dependencies are a key factor in database design. Large amounts of literature have been devoted to this topic, starting with Codd's pioneer work ([5]), and a complete axiomatization by Fagin, Beeri and Howard [10]. The most well known handbooks on database design also deepen into the underlying theory but from a more practical point of view ([18], [6], [7]). Although being a classical topic, functional dependencies in databases is still an active research topic, mainly in its algorithmic and complexity aspect ([14], [15], [4]), and strongly connected to learning theory [13]. A functional dependency describes the relationship between two sets of attributes with a rule $X \to Y$, that states that the values of the set of attributes X

*This work is supported in part by EU ESPRIT IST-1999-14186 (ALCOM-FT), MCYT TIC2002-04019-C03-01 (Moises), and matching funds TIC2000-1970-CE, and CIRIT 2001SGR-00252.

determine the value of the set of attributes Y. These dependencies are extremely helpful to a database designer in order to know if a database is in third Normal Form.

Moreover, the authors of [8] and [9], discovered an equivalence between functional dependencies and a fragment of propositional logic. In fact, the main issue was the equivalence between the inference rules that are to be used in order to derive more functional dependencies out of a given set of functional dependencies, and the axioms that are used to form new implications in propositional calculus.

In the first section of this paper, a formal description of a concept lattice will be provided, as well as a formalization of a functional dependency, just in order to standarize the notation that will be used along the paper. In the second section, a new closure operator will be presented, and some of its properties will be reviewed. In the third section, it will be proved that, given a concrete, specific closure operator, concept lattices can determine the functional dependencies in a database. In the fourth section, it will be proved that this operator generalizes the operator that has been used to find association rules in a database.

2 Definitions

2.1 Concept Lattices

Let $\langle O, I, R\rangle$ be a formal context. This means that O is a set of objects, I is a set of attributes, and $R \subseteq O \times I$ is a binary relation between sets O and I. This formal context can be viewed as a database, as a set of observations, or as a set of models of a logical formula, among many other forms. From now on, we will use the letters X, Y, Z to denote sets of attributes, x, y, z to denote single attributes, A, B, C to denote sets of objects, and a, b, c to denote single objects. Let ϕ, ψ be two operators such that:

- $\phi : \mathcal{P}(I) \rightarrow \mathcal{P}(O)$. This operator returns the objects that have a relation with all the attributes of the given set of attributes, it is, $\phi(X) = \{a \in O | \langle a, x\rangle \in R, \forall x \in X\}$.
- $\psi : \mathcal{P}(O) \rightarrow \mathcal{P}(I)$. Given a set of objects, this operator returns the largest set of attributes such that all the objects have a relationship with all the attributes, it is, $\psi(A) = \{x \in I | \langle a, x\rangle \in R, \forall a \in A\}$.

Both operators follow a set of properties (for a proof for each property, please refer to [11]):

- $X \subseteq Y \Rightarrow \phi(Y) \subseteq \phi(X)$
- $X \subseteq \psi(\phi(X))$
- $\phi(X) = \phi(\psi(\phi(X)))$
- $A \subseteq B \Rightarrow \psi(B) \subseteq \psi(A)$
- $A \subseteq \phi(\psi(A))$
- $\psi(A) = \psi(\phi(\psi(A)))$
- $X \subseteq \psi(A) \Leftrightarrow A \subseteq \phi(X)$

The pair (ϕ, ψ) forms what is called a *Galois connection*, which is a set of two operators having the following property:

- $A \leq \phi(X) \Leftrightarrow X \leq \psi(A)$. (different sets of equivalent properties can be found in [11]).

The fact that these two operators form a Galois connection, implies that their composition ($\psi.\phi$ and $\phi.\psi$) forms a *closure operator* (for a proof, please refer to [11]). A closure operator (φ) has the following properties:

- $X \subseteq \varphi(X)$
- $X \subseteq Y \rightarrow \varphi(X) \subseteq \varphi(Y)$
- $\varphi(X) = \varphi(\varphi(X))$

The operator $\varphi = \psi.\phi$ will be used to induce a partition of all the possible sets of attributes in a relation in classes. The closure of a set of attributes X will be $\varphi(X)$. A *formal concept* of (O, I, R) is a pair $\langle X, A\rangle$ such that $A \in O$ and $X \in I$, and $\phi(X) = A$ and $\psi(A) = X$, and let T be the set of all formal concepts. A precise definition of a *concept lattice* can now be given: let O be a set of objects, I a set of attributes, and $R \subseteq O \times I$ a binary relation, let $\varphi = \psi.\phi$ be a closure operator as previously defined, and let B the set of all concepts of (O, I, R), a *concept lattice* is the the set B plus an infimum and a supremum given by

$$\bigwedge_{t\in T}(X_i, A_i) = \left(\bigcap_{t\in T} X_t, \phi(\psi(\bigcup_{t\in T} A_t))\right), \bigvee_{t\in T}(X_i, A_i) = \left(\psi(\phi(\bigcap_{t\in T} X_t)), \bigcap_{t\in T} A_t\right)$$

where $X \in \mathcal{P}(I)$ and $A \in \mathcal{P}(O)$. In practice, the graphical concept lattice is represented as follows: the nodes (concepts) are the sets of attributes $\{X|X = \varphi(X)\}$, and the infimum is the set of attributes that has a relation with all the objects, it is, the set of attributes that appear in all the tuples (which can be eventualy empty), and the supremum is the set of all the attributes. A node X will be connected by a line to another node Y if it immediately includes or it is immediately included by Y. For practical reasons, the sets of objects have been erased from the concepts.

The set of generators of a closure X is $\Gamma(X) = \{Y|\varphi(Y) = X, \nexists Y' \subset Y : \varphi(Y') = X\}$, which, together with its closure can form *inference rules*, of the kind $Y \rightarrow X \setminus Y$ where $Y \in \Gamma(X)$. These rules have mainly been used to find relations of the kind "if X appears, then Y also appears". The concept lattice thus formed explains what attributes depend on other attributes ([17]).

The main limitation of this model is the fact that the tuples are binary, and not multivalued, and so, it covers a broad yet limited set of possible cases. In this paper this limitation will be overcome, since we will be dealing with multivalued databases, it is, databases in which the set of values that a relation can take is not limited to a binary domain.

2.2 Functional Dependencies

Many literature has been devoted to functional dependencies. The reader can refer to [10] to find a complete and formal explanation. For the purposes of this paper, we will define the following elements: let I be a finite set of attributes, let O be a finite set of objects (tuples), and let V be a finite set of possible values an attribute can take, and relation R (informaly, a database) is a subset of $I \times O \times V$. For a given $a \in O$, $a[X]$ is the ordered set of values that the object a takes for each $x \in X$ (the order in the values will be according to an order induced in the set I).

A functional dependency $X \rightarrow y$, (where X is a set of attributes, and y is a single attribute) *holds* in a relation R if for all pair of tuples (objects) $a_i, a_j : a_i[X] = a_j[X] \rightarrow a_i[y] = a_j[y]$, it is, the set of attributes X determines the value of the attribute y. A functional dependency is *minimal* if no dependency with a proper subset of X as antecedent and the same consequent holds.

In [14] the following notation has been presented: a set of attributes can *induce* a partition of the tuples in classes, according to the values of the attributes, in which each class or subset in this partition will contain those tuples that have the same value for a given set of attributes X. A more formal definition of a partition Π_X is: $\Pi_X = \{P_i|\forall a_i, a_j \in P_i, a_i[X] = a_j[X]\}$, It follows that $\forall P_i, P_j \in \Pi_X, P_i \cap P_j = \emptyset$ and that $\bigcup P_i = O$, it is, all the sets in a given partition are disjoint, and complete. The number of classes in Π_X is $|\Pi_X|$.

A partition Π_Y *refines* a partition Π_X iif $\forall P_i \in \Pi_Y \exists P_j \in \Pi_X P_i \subseteq P_j$. It then follows that if Π_Y refines Π_X, then $|\Pi_X| \leq |\Pi_Y|$. Following the partition induced by a set of attributes, the following theorem relates functional dependencies and partitions:

Theorem 2.1 *A functional dependency $X \rightarrow A$ holds iif $|\Pi_X| = |\Pi_{X\cup A}|$*

(See [14] for a proof).

3 Concept lattices as a framework to find functional dependencies

Our goal will be to provide a framework based on concept lattices that will enable us to find all minimal functional dependencies. To do so, we will provide a new Galois connection, whose composition will form a new closure operator. Using this operator, a concept lattice will be created, and then, it will be proved that the set of rules that will be extracted out of this concept lattice will be equivalent to the set of minimal functional dependencies in that database. The new operator will be able to deal with multivalued databases.

It should be noted that Ganter and Wille [11] developed a method to extract the functional dependencies out of a multivalued database. The method transformed the original database DB in a new binary database DB', and then, the current analysis proceeded. The transformation basicaly consisted in intersecting all tuples with each other, and for each pair of records $r_i, r_j \in DB$, a new record $s_i \in DB'$ is created in the new database, and for each single attribute x, $s_i[x] = 1$ if $r_i[x] = r_j[x]$, zero otherwise. It is then proved that applying the current analysis to this newly created database, the resulting association rules are functional dependencies in DB. Instead of transforming the database, our aim will be to create a new closure operator.

3.1 A new closure operator

$\Pi_{\mathcal{P}(I)}$ is the class of all possible partitions that can be induced by any set of attributes. We define the following operators:

- Φ: $\mathcal{P}(I) \rightarrow \Pi_{\mathcal{P}(I)}$. Given a set of attributes $X \subseteq I$, it returns the partition induced by X: Π_X, it is, a set of classes.
- Ψ: $\Pi_{\mathcal{P}(I)} \rightarrow \mathcal{P}(I)$. Given a partition Π_X, it returns a set of attributes X' such that $\Pi_X = \Pi_{X'}$ and $\nexists X'' : X' \subset X'', \Pi_{X'} = \Pi_{X''}$. It is: it returns the largest set of attributes that can induce the given partition. It is worth mentioning two facts: first, this operator can eventualy be undefined, since there can be no way to partition a given set of tuples according to any set of attributes. For instance, it is not possible to partition a set of tuples into only one class, if there are no attributes that have the same value for all the tuples in the database. It follows that this operator should be carefully used in order to avoid this possibility. Second, if this set exists and is the largest one, it is unique. It can be easily proved that if two different sets X, Y have these properties (induce the partition and are the largest), then $X \cup Y$ also has the same property, and it is larger than any of both of them.

We also define an order for attributes and an order for partitions.

1. $(\mathcal{P}(I), \leq)$, where $X, Y \subseteq I$: $X \leq Y$ iif $X \subseteq Y$.
2. $(\Pi_{\mathcal{P}(I)}, \preceq)$, where $\Pi_X, \Pi_Y \subseteq \Pi_{\mathcal{P}(I)}$: $\Pi_X \preceq \Pi_Y$ iif Π_X refines Π_Y ($|\Pi_X| \geq |\Pi_Y|$).

In order to prove that these operators form a Galois connection, we need the to show different propositions:

Proposition 3.1 $Y \subseteq X \rightarrow \Pi_X$ *refines* Π_Y

Proof: If we suppose that it is not true, then Π_X does not refine Π_Y. It means that at least there will be two objects a_1, a_2 that will belong to two different classes in Π_Y and to the same class on Π_X. Then, $a_1[X] = a_2[X]$. And if they belong to different classes in Π_Y, it means that $\exists Z \subseteq Y$ such that $a_1[Z] \neq a_2[Z]$. Therefore, $Z \subseteq Y$ and $Z \nsubseteq X$, but it contradicts the previous assumption $Y \subseteq X$.

Proposition 3.2 *If $X, Y \subseteq I$, are both the maximum sets of attributes that induce the classes Π_X, Π_Y and if Π_X refines Π_Y, then, $Y \subseteq X$.*

Proof: Let P_Y be a class in Π_Y, and let P_X be a class in Π_X such that $P_X \subseteq P_Y$. Let us suppose that that $X \subset Y$. It implies that $\exists Z \in Y$ such that $Z \notin X$. Since P_X belongs to a class, it means that $\forall a_1, a_2 \in P_X, a_1[X] = a_2[X]$, and since X is the maximum class that induces Π_X, it follows that there is no other attribute Z which is not included in X such that $a_1[X \cup Z] = a_2[X \cup Z]$. It contradicts our previous assumption $X \subset Y$ and $\forall a_1, a_2 \in P_Y, a_1[Y] = a_2[Y]$

Proposition 3.3 *Given a set of attributes X and $y \in I$, if $|\Pi_X| = |\Pi_{X\cup\{y\}}|$, then, $\Pi_X = \Pi_{X\cup\{y\}}$*

Proof: Given a partition Π_X, if we add an attribute y to X, the partition $\Pi_{X\cup\{y\}}$ will be a refinement of Π_X. Since $|\Pi_X| = |\Pi_{X\cup\{y\}}|$, it follows that all the classes in Π_X must be the same as in $\Pi_{X\cup\{y\}}$.

Proposition 3.4 $|\Pi_X| \leq |\Pi_{X\cup\{y\}}|$.

Proof: Let us suppose that $|\Pi_X| > |\Pi_{X\cup\{y\}}|$. There will be two elements a_1, a_2 that are in two different classes in Π_X, but that will be in the same class in $\Pi_{X\cup\{y\}}$, which means that they will have the same value for the attributes $X \cup \{y\}$. But this contradicts the previous assumption that they were in different classes in Π_X.

We are now able to prove that the previously defined operators form a Galois connection.

Proposition 3.5 *$(\Phi, \leq)$ and $(\Psi, \preceq)$, form a Galois Connection.*

Given $X_1, X_2, Y \subseteq I$ and $A_1, A_2, B \in \Pi_{\mathcal{P}(I)}$, the following propositions must be proved:

1. $x_1 \leq x_2 \rightarrow \Phi(X_1) \succeq \Phi(X_2)$. Proof: Easily by proposition [3.1].
2. $A_1 \preceq A_2 \rightarrow \Psi(A_1) \geq \Psi(A_2)$. Proof: Easily by proposition [3.2].
3. $Y \leq \Psi(\Phi(Y))$. Proof: $\Phi(Y) = \Pi_Y$, $\Psi(\Pi_Y) = Z$, being Z the greatest set such that $\Pi_Z = \Pi_Y$. Since Z is the greatest, it follows that $Z \geq Y$.
4. $P \preceq \Phi(\Psi(B))$. Proof: $\Psi(B) = X$, being X the greatest set such that $\Pi_X = P$, and then, $\Phi(X) = \Pi_X = P$.

The fact that these operators form a Galois connection, enables us to prove that their composition is a closure operator.

Proposition 3.6 *Since Φ and Ψ form a Galois Connection, $\Gamma = \Psi.\Phi$, is a closure operator.*

The proof is given in [11] (proposition 8).

4 Concept lattices and functional dependencies

According to the newly created closure operator $\Gamma = \Phi.\Psi$, we can generate a closure lattice, and then form the rules out of each closed set and its generators. We claim that:

Theorem 4.1 *The set of rules of a concept lattice formed by the closure operator Γ is logically equivalent to the set of minimal functional dependencies (both sets are logically implied by each other).*

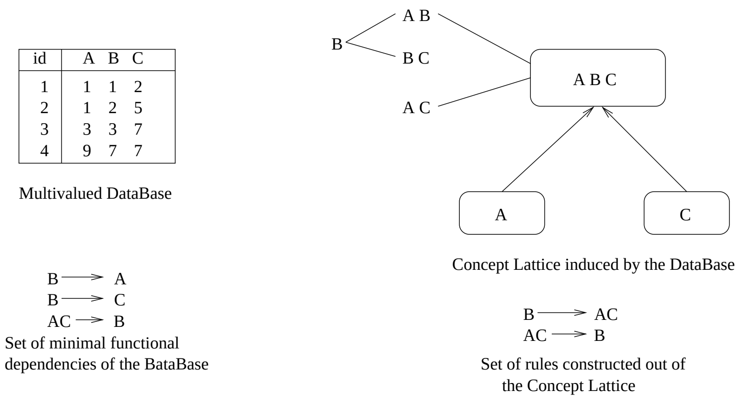
Proof:

$\Rightarrow$) Any minimal functional dependency will be implied by a rule from the concept lattice formed by the closure operator Γ: Let $X \rightarrow y$ be a functional dependency. Then, $|\Pi_X| = |\Pi_{X\cup\{y\}}|$, and, by [3.3], $\Pi_X = \Pi_{X\cup\{y\}}$. Since the partitions induced by both X and $X \cup \{y\}$ are the same, they belong to the same closure (by construction of Γ). Let $Z = \Gamma(X) = \Gamma(X \cup \{y\})$. Since $X \rightarrow y$ is minimal, $\nexists X' \subset X$ such that $|\Pi'_X| = |\Pi_{X'\cup\{y\}}|$, which means that X is a generator of

Z. By construction of the concept lattice, the following rule will be formed: $X \rightarrow Z \setminus X$. Since $y \in Z \setminus X$, this rule in the concept lattice will imply the functional dependency.

$\Leftarrow$) Any rule in the concept lattice will be implied by a (set of) functional dependency(ies): Let X be a minimal generator of a concept $\Gamma(X)$. Then, the rule $X \rightarrow \Gamma(X) \setminus X$ will be formed by construction of the Concept Lattice. Let $\Gamma(X) \setminus X = Z$. Since X is a generator, then $\nexists X' \subset X$ such that $\Gamma(X') = \Gamma(X)$. Therefore, if the generated rule is a functional dependency, it will be minimal.

Let $Z = \{z_1, z_2, \ldots, z_n\}$, then we need to prove that the functional dependencies $X \rightarrow z_1, X \rightarrow z_2 \ldots, X \rightarrow z_n$ will hold, since $X \rightarrow z_1, X \rightarrow z_2 \ldots, X \rightarrow z_n \models X \rightarrow Z$. The proof is for a given $X \rightarrow z_i$. Let us suppose that $X \rightarrow z_i$ does not hold: it means that $|\Pi_X| < |\Pi_{X \cup \{z_i\}}|$. But we know that $|\Pi_X| = |\Pi_{Z \cup X}|$, and that since $z_i \in Z$, then by [3.1] $|\Pi_{X \cup \{z_i\}}| \leq |\Pi_{X \cup Z}|$, which is a contradiction. Analogous reasoning can be stated for all $z_i \in Z$.



id	A	B	C
1	1	1	2
2	1	2	5
3	3	3	7
4	9	7	7

Multivalued DataBase
B
A B
B C
A C
A B C
A
C
Concept Lattice induced by the DataBase
B → A
B → C
AC → B
Set of minimal functional dependencies of the BataBase
B → AC
AC → B
Set of rules constructed out of the Concept Lattice

Example of how the proposed closure operator works

5 Generalization of the finding of association rules

With the next theorem, it will be proved that the closure operator defined in the previous section generalizes the closure operator φ which has been presented in section 2.1, and thus, we will able to affirm that in fact, the analysis of what has been performed with concept lattices is a subcase of the finding of functional dependencies in a database. Since in the classical approach we have been dealing with binary relations (databases), and in the new approach we are dealing with multivalued relations, we need a function to *translate* this difference, which is the following:

Definition 5.1 *For partitions of objects over the domain $V = \{0,1\}$, let $f : \Pi_{\mathcal{P}(I)} \rightarrow \Pi_{\mathcal{P}(I)}$ be a function such that given Π_X returns a set $P_i \in \Pi_X$ such that $\forall o \in P_i, o[X] = \{11\ldots1\}$, or the empty set otherwise.*

We are now ready to see how our method generalizes the former one:

Theorem 5.2 *The operator $\Gamma' = \Psi.f.\Phi$ creates the same set of rules out of a set of models that the operator defined in current concept lattice literature (φ).*

Proof: We only need to prove that $\forall X \subseteq I : \varphi(X) = \Gamma'(X)$. Let $\varphi = \psi.\phi$. By construction, $\phi(X)$ is the collection of objects such that $\forall o \in \phi(X), o[X] = \{11\ldots1\}$. And $\psi(\phi(X))$ is the largest set of objects such that complies with the former condition.

On the other hand, we have that $\Phi(X)$ is the partition induced by $X = x_1, \ldots x_n$. The function f filters all the sets of objects, and retains the one such that $\forall o \in x_i, x_i[X] = \{111\ldots1\}$. Since all possible values are $\{0,1\}$, this possibility holds. Then, $\Psi(f(\Phi(X)))$ returns the largest set of attributes that induces a set of objects such that $\forall o \in \phi(X), o[X] = \{111\ldots1\}$.

References

[1] Balcázar, J.L., Baixeries J. *Discrete Deterministic Data Mining as Knowledge Compilation.* Workshop on Discrete Mathematics and Data Mining in SIAM International Conference on Data Mining (2003).

[2] Bastide Y., Pasquier N., Taouil R., Stumme G., Lakhal L. *Mining Minimal Non-Redundant Association Rules using Closed Itemsets.* Proc. of the 1st Int'l Conf. on Computational Logic, num. 1861, Lectures Notes in Artificial Intelligence, Springer, july 2000, pages 972-986.

[3] Birkhoff G. *Lattice Theory*, first edition. Amer. Math. Soc. Coll. Publ. 25, Providence, R.I. 1973.

[4] Castellanos M. and Saltor F. *Extraction of Data Dependencies.* Information Modelling and Knowledge Bases V. IOS Press, Amsterdam, 1994, pp. 400-420.

[5] Codd, E. F. *A relational model for large shared database banks.* Communications ACM 16, 6 (June 1970), 377 - 387.

[6] Date C.J. *Introduction to Database Systems.* Addison-Wesley, 7 edition, 2000.

[7] Elmasri R., Navathe S. B. *Fundamentals of Database Systems.* 2nd Edition. Benjamin/Cummings 1994

[8] Fagin R. *Functional dependencies in a relational database and propositional logic.* IBM J. Research and Development 21, 6, Nov. 1977, pp. 534-544.

[9] Fagin R., Sagiv Y., Delobel D., and Stott Parker D. *An equivalence between relational database dependencies and a fragment of propositional logic.* Jr. J. ACM 28, 3, July 1981, pp. 435-453. Corrigendum: J. ACM 34, 4, Oct. 1987, pp. 1016-1018.

[10] Fagin R., Beeri C., Howard J. H. *A complete axiomatization for functional and multivalued dependencies in database relations.* Jr. Proc. 1977 ACM SIGMOD Symposium, ed. D. C. P. Smith, Toronto, pp. 47-61.

[11] Ganter, B., Wille R. *Formal Concept Analysis. Mathematical Foundations.* Springer, 1999.

[12] Godin, R. and Missaoui, R. *An Incremental Concept Formation Approach for Learning from Databases.* Theoretical Computer Science, Special Issue on Formal Methods in Databases and Software Engineering, 133, 387-419.

[13] Hermo M. and Lavín V. *Learning Minimal Covers of Functional Dependencies with Queries.* Lecture Notes in Artificial Intelligence. Proceedings of ALT'99, Vol. 1720, Springer (1999).

[14] Huhtala Y., Karkkainen J., Porkka P., Toivonen H. *TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies.* The Computer Journal 42(2): 100 - 111, 1999.

[15] Kivinen J. and Mannila H. *Approximate inference of functional dependencies from relations.* Theoretical Computer Science 149(1) (1995), 129-149.

[16] Pfaltz, J.L. *Transformations of Concept Graphs: An Approach to Empirical Induction.* 2nd International Workshop on Graph Transformation and Visual Modeling Techniques. GTVM 2001, Satellite Workshop of ICALP 2001, Crete, Greece. Pages 320-326. July 2001.

[17] Pfaltz, J.L., Taylor, C.M. *Scientific Discovery through Iterative Transformations of Concept Lattices.* Workshop on Discrete Mathematics and Data Mining at 2nd SIAM Conference on Data Mining, Arlington. Pages 65-74. April 2002.

[18] Ullman J.D. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, Inc. 1988.

[19] Valtchev P., Missaoui R. *Building Galois (Concept) Lattices from Parts: Generalizing the Incremental Approach*. Proceedings of the ICCS 2001, LNCS 2120, Springer Verlag, pp. 290-303, Stanford (CA), 2001.