On Monotone Data Mining Languages

Toon Calders[1] and Jef Wijsen[2]

[1] Research Assistant of the Fund for Scientific Research – Flanders (Belgium) (F.W.O. – Vlaanderen), University of Antwerp
calders@uia.ua.ac.be
[2] University of Mons-Hainaut, Belgium
jef.wijsen@umh.ac.be

Abstract. We present a simple Data Mining Logic (DML) that can express common data mining tasks, like "Find Boolean association rules" or "Find inclusion dependencies." At the center of the paper is the problem of characterizing DML queries that are amenable to the levelwise search strategy used in the a-priori algorithm. We relate the problem to that of characterizing monotone first-order properties for finite models.

1 Introduction

In recent years, the problem of finding frequent itemsets in market-basket data has become a popular research topic. The input of the problem is a database storing baskets of items bought together by customers. The problem is to find sets of items that appear together in at least $s\%$ of the baskets, where s is some fixed threshold; such sets are called *frequent itemsets*. Although the problem of finding frequent itemsets can easily be stated as a graph-theoretical problem, the formulation in marketing terms [1] probably contributed much to the success of the problem.

The *a-priori* algorithm is probably the best-known procedure to solve this problem. It is based on a very simple property: If a set X of items is no frequent itemset, then no superset of X is a frequent itemset either. This property has been given different names; in [4, page 231] it is called *anti-monotone*, and defined as: "If a set cannot pass a test, all of its supersets will fail the same test as well." The *a-priori* algorithm thus first searches for singleton frequent itemsets, and then iteratively evaluates ever larger sets, while ignoring any set that cannot be frequent because a subset of it turned out to be infrequent in earlier iterations. The anti-monotonicity property underlying the *a-priori* algorithm has subsequently been generalized to *levelwise search* [11]. As a matter of fact, the a-priori trick is applicable in many other data mining tasks, such as the discovery of keys, inclusion dependencies, functional dependencies, episodes [10, 11], and other kinds of rules [16]. With the advent of data mining primitives in query languages, it is interesting and important to explore to which extent the a-priori technique can be incorporated into next-generation query optimizers.

During an invited tutorial at ICDT'97, Heikki Mannila raised an interesting and important research problem:

> "What is the relationship between the logical form of sentences to be discovered and the computational complexity of the discovery task?" [10, slide 51]

It is natural to ask a related question about the relationship between the logical form of sentences and the applicability of a given data mining technique, like the a-priori technique:

> "What is the relationship between the logical form of sentences to be discovered and the applicability of a given data mining technique?"

This question is of great importance when we move to database systems that support data mining queries. Data mining querying differs from standard querying in several respects [5], and conventional optimizers that were built for standard queries, may not perform well on data mining queries. Next-generation query optimizers must be able to decide which data mining optimization techniques are effective for a given data mining query. In the domain of mining frequent itemsets and association rules, there has been a number of recent papers relating to the second question raised above. Lakshmanan et al. [7, 12] have introduced the paradigm of constrained frequent set queries. They point out that users typically want to impose constraints on the itemsets to be discovered (for example, itemsets must contain *milk*); they then explore the relationships between the properties of constraints on the one hand and the effectiveness of certain pruning optimizations on the other hand. Tsur et al. [15] explore the question of how techniques like the a-priori algorithm can be generalized to parameterized queries with a filter condition, called query flocks. In spite of these works, it seems fair to say that the relationship between the form of sentences to be discovered and the applicability of data mining techniques has not been systematically explored, and that a clean unifying framework is currently missing.

In this paper, we further explore from a logic perspective the relationship between the form of sentences to be discovered and the applicability of the a-priori technique. To this extent, we first have to decide upon which logic to use. The logic should allow expressing some basic rule (or dependency) mining tasks, like mining Boolean association rules, functional dependencies, or inclusion dependencies. As dependencies are mostly stated in terms of attributes, we propose a logic, called Data Mining Logic (DML), that extends relational tuple calculus with variables ranging over attributes and over sets of attributes (i.e., over relational schemas). We do not claim originality for the DML way of querying schemas; in fact, variables ranging over attribute and relation names also appear in other languages [8, 13]. Our main objective was not to design a new language, however, but rather to answer the question of which classes of queries are amenable to levelwise search. DML provides an adequate framework for exploring that question. Moreover, we believe that the generality of the language allows "transplanting" the results in other frameworks. The main contribution of the paper lies in revealing a significant relationship between the applicability of the a-priori technique in DML queries and monotone first-order properties for finite models.

The paper is organized as follows. Section 2 illustrates DML by an example. The syntax and semantics of DML are defined in Section 3. In Section 4, we show how certain common data mining tasks can be expressed in DML. Section 5 introduces subset-closed and superset-closed queries; these are the queries that admit levelwise search. Unfortunately, these query classes are not recursive. Section 6 introduces a recursive subclass of superset-closed queries, called positive queries. Although many "practical" superset-closed queries can be expressed positively, the class of positive queries does not semantically cover the whole class of superset-closed queries. The latter result is proved in Sections 7 through 9. Finally, Section 10 concludes the paper. Detailed proofs of all lemmas and theorems can be found in [3].

2 Introductory Example

We extend the relational tuple calculus with *attribute-variables* that range over attributes, and *schema-variables* that range over sets of n-ary tuples of attributes. In the following example, X is an attribute-variable and $\mathcal{X}$ a unary schema-variable. The query asks for sets $\mathcal{X}$ of attributes such that at least two distinct tuples t and s agree on each attribute of $\mathcal{X}$. Requiring that t and s be distinct is tantamount to saying that they disagree on at least one attribute Z.

$$\{\mathcal{X} \mid \exists t(\exists s(\neg\exists Y(\mathcal{X}(Y) \wedge t.Y \neq s.Y) \wedge \exists Z(t.Z \neq s.Z)))\} \tag{1}$$

For the relation:

R	A	B	C	D
	0	0	0	1
	0	0	1	2
	0	1	1	3
	1	1	2	2

the result is:

$\mathcal{X}^{(1)}$
$\{A, B\}$
$\{A, C\}$
$\{A\}$
$\{B\}$
$\{C\}$
$\{D\}$
$\{\}$

.

Note that $\{B, C\}$ is not in the answer set, since R does not contain two distinct tuples that agree on both B and C. One can easily verify that whenever a set appears in the above result then all its subsets appear as well. Moreover, this property remains true no matter what is the schema or the content of the input relation R. We will say that the query is *subset-closed*.

3 DML Syntax and Semantics

3.1 Syntax

We define our Data Mining Logic (DML). The idea is to extend relational tuple calculus with *attribute-variables* that range over attributes, and *schema-variables* that range over sets of n-ary tuples of attributes. For simplicity, we assume that the database consists of a single relation; therefore there is no need to introduce predicate symbols.

DML Alphabet

- Denumerably many *attribute-variables* $X, Y, Z, X_1, Y_1, Z_1, \ldots$
- Denumerably many *tuple-variables* $t, s, t_1, s_1, \ldots$
- For every $n \in \mathbb{N}$, at most denumerably many *n-ary schema-variables* $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{X}_1$, $\mathcal{Y}_1$,...
- A set $\mathbf{C}$ of *constants*.

For simplicity, the arity of a schema-variable may be denoted in superscript: $\mathcal{X}^{(n)}$ denotes an n-ary schema-variable. Attribute-variables and tuple-variables together are called *simple-variables*.

Atomic DML Formulas

1. If X and Y are attribute-variables, t and s are tuple-variables, and a is a constant, then $X = Y$, $t.X = s.Y$, and $t.X = a$ are atomic DML formulas.
2. If $\mathcal{X}$ is an n-ary schema-variable, and $X_1, \ldots, X_n$ are attribute-variables, then $\mathcal{X}(X_1, \ldots, X_n)$ is an atomic DML formula.

DML Formulas

1. Every atomic DML formula is a DML formula.
2. If δ_1 and δ_2 are DML formulas, then $\neg\delta_1$ and $(\delta_1 \vee \delta_2)$ are DML formulas.
3. If δ is a DML formula and X is an attribute-variable, then $\exists X(\delta)$ is a DML formula.
4. If δ is a DML formula and t is a tuple-variable, then $\exists t(\delta)$ is a DML formula.

Note that the existential quantifier $\exists$ can be followed by an attribute-variable as well as a tuple-variable. These two usages of $\exists$ will have different semantics. Since attribute-variables and tuple-variables are assumed to be distinct, the double use of $\exists$ does not result in any confusion.

A DML formula is called *closed* iff all occurrences of simple-variables (i.e., tuple-variables and attribute-variables) are bound, where boundedness is defined as usual. A closed DML formula is also called a *DML sentence*. The abbreviations $\wedge, \rightarrow, \leftrightarrow, \mathbf{true}, \mathbf{false}, \forall, \neq$, with conventional precedence relationship, are introduced as usual. In addition we introduce the abbreviations:

- $\forall\mathcal{X}(X_1, \ldots, X_n)(\delta)$ for $\forall X_1(\ldots(\forall X_n(\mathcal{X}(X_1, \ldots, X_n) \rightarrow (\delta)))\ldots)$, and
- $t = s$ for $\forall X(t.X = s.X)$.

DML Queries A *DML query* is an expression of the form $\{\mathcal{X}_1, \ldots, \mathcal{X}_m \mid \delta\}$, where δ is a DML sentence and $\mathcal{X}_1, \ldots, \mathcal{X}_m$ are exactly all distinct schema-variables occurring in δ ($m \geq 1$).

3.2 DML Semantics

We assume the existence of a set **att** of *attributes*. A *schema* is a finite, nonempty set of attributes.[3] A *tuple* over the schema S is a total function from S to the set **C** of constants. A *relation* is a finite set of tuples.

The notion of *DML structure* is defined relative to a schema S: It is a pair $\langle R, \Sigma\rangle$ where R is a relation over S and Σ is a *schema-variable assignment* assigning some $\Sigma(\mathcal{X}^{(n)}) \subseteq 2^{(S^n)}$ to every n-ary schema-variable $\mathcal{X}^{(n)}$. For convenience, the schema of R will be denoted $|R|$.

A *DML interpretation* is a pair $\langle\langle R, \Sigma\rangle, \sigma\rangle$ where $\langle R, \Sigma\rangle$ is a DML structure and σ is a *simple-variable assignment* assigning some tuple over $|R|$ (which may or may not belong to R) to every tuple-variable t, and assigning some $\sigma(X) \in |R|$ to every attribute-variable X.

The *satisfaction of DML formulas* is defined relative to a DML interpretation $\langle\langle R, \Sigma\rangle, \sigma\rangle$: [4]

$$
\begin{array}{ll}
\langle\langle R, \Sigma\rangle, \sigma\rangle \models X = Y & \text{iff } \sigma(X) = \sigma(Y) \\
\langle\langle R, \Sigma\rangle, \sigma\rangle \models t.X = s.Y & \text{iff } \sigma(t)(\sigma(X)) = \sigma(s)(\sigma(Y)) \\
\langle\langle R, \Sigma\rangle, \sigma\rangle \models t.X = a & \text{iff } \sigma(t)(\sigma(X)) = a \\
\langle\langle R, \Sigma\rangle, \sigma\rangle \models \mathcal{X}(X_1, \ldots, X_n) & \text{iff } (\sigma(X_1), \ldots, \sigma(X_n)) \in \Sigma(\mathcal{X}) \\
\langle\langle R, \Sigma\rangle, \sigma\rangle \models \neg\delta & \text{iff } \langle\langle R, \Sigma\rangle, \sigma\rangle \not\models \delta \\
\langle\langle R, \Sigma\rangle, \sigma\rangle \models \delta_1 \vee \delta_2 & \text{iff } \langle\langle R, \Sigma\rangle, \sigma\rangle \models \delta_1 \text{ or } \langle\langle R, \Sigma\rangle, \sigma\rangle \models \delta_2 \\
\langle\langle R, \Sigma\rangle, \sigma\rangle \models \exists X(\delta) & \text{iff } \langle\langle R, \Sigma\rangle, \sigma_{X\to A}\rangle \models \delta \text{ for some } A \in |R| \\
\langle\langle R, \Sigma\rangle, \sigma\rangle \models \exists t(\delta) & \text{iff } \langle\langle R, \Sigma\rangle, \sigma_{t\to r}\rangle \models \delta \text{ for some } r \in R
\end{array}
$$

As mentioned before, we use only a single relation R to simplify the notation. Importantly, the semantics specifies that the quantification $\exists X(\delta)$, where X is an attribute-variable, is over the (finite) schema $|R|$ of R. Also, the quantification $\exists t(\delta)$, where t is a tuple-variable, is over the (finite) relation R. For example, the statement $\exists t(\exists X(t.X = 1))$ is satisfied relative to a DML interpretation $\langle\langle R, \Sigma\rangle, \sigma\rangle$ if the value 1 occurs somewhere in the relation R. The statement $\exists t(\mathbf{true})$ is satisfied if R contains at least one tuple.

The satisfaction of closed DML formulas does not depend on the simple-variable assignment σ. We write $\langle R, \Sigma\rangle \models \delta$ iff $\langle\langle R, \Sigma\rangle, \sigma\rangle \models \delta$ for every simple-variable assignment σ.

The answer to a DML query is defined relative to a relation R. The answer to the DML query $\{\mathcal{X}_1, \ldots, \mathcal{X}_m \mid \delta\}$ is the set:

$$\{(\Sigma(\mathcal{X}_1), \ldots, \Sigma(\mathcal{X}_m)) \mid \langle R, \Sigma\rangle \text{ is a DML structure satisfying } \delta\} \ .$$

[3] The extension to schemas without attributes is possible but less pertinent in a data mining context.

[4] If f is a function then $f_{x\to a}$ is the function satisfying $f_{x\to a}(y) = f(y)$ for every y other than x, and $f_{x\to a}(x) = a$. $f_{x\to a, y\to b}$ is a shorthand for $(f_{x\to a})_{y\to b}$.

4 Additional Examples

4.1 Frequent Itemsets and Non-trivial Functional Dependencies

Consider a relation where every attribute represents a product, and every tuple a customer transaction. For a given row t and attribute A, the value $t(A) = 1$ if the product A was bought in the transaction t, and $t(A) = 0$ otherwise. Next, to be able to store two transactions containing exactly the same products, a special attribute *TID* is needed that serves as the unique transaction identifier. We assume that the values 0 and 1 are not used to identify transactions, so that *TID* cannot possibly be interpreted as a product. The data mining problem is to find frequent itemsets [1]: Find sets $\mathcal{X}$ of attributes such that at least n distinct tuples have value = 1 for all attributes of $\mathcal{X}$. The value $n > 0$ is an absolute support threshold in this example. The DML query is as follows:

$$\{\mathcal{X} \mid \exists t_1, \ldots, t_n(\bigwedge_{1 \leq i < j \leq n} t_i \neq t_j \wedge \forall \mathcal{X}(Y)(t_1.Y = 1 \wedge \ldots \wedge t_n.Y = 1))\} \quad (2)$$

The following DML query asks for non-trivial functional dependencies, i.e., functional dependencies whose right-hand side is not a subset of the left-hand side:

$$\begin{aligned}\{\mathcal{X}, \mathcal{Y} \mid \forall t(\forall s((\forall \mathcal{X}(X)(t.X = s.X)) \rightarrow (\forall \mathcal{Y}(Y)(t.Y = s.Y)))) \\ \wedge \exists \mathcal{Y}(Y)(\neg \mathcal{X}(Y))\}\end{aligned} \quad (3)$$

For the relation:

R	A	B
	0	0
	0	1
	1	2

the result is:

$\mathcal{X}$	$\mathcal{Y}$
$\{B\}$	$\{A\}$
$\{B\}$	$\{A, B\}$

.

The two lines of the result encode the functional dependencies $\{B\} \rightarrow \{A\}$ and $\{B\} \rightarrow \{A, B\}$ respectively. The discovery of functional dependencies has been studied for many years now (see for example [6, 9]).

4.2 The Use of Binary Schema-variables: Inclusion Dependencies

In all examples introduced so far, all schema-variables were unary, i.e., had arity 1. We now illustrate the need for binary schema-variables. In the example, sets of attribute pairs are used to encode inclusion dependencies that hold in a single relation. An inclusion dependency $\langle A_1, \ldots, A_n \rangle \subseteq \langle B_1, \ldots, B_n \rangle$ can be encoded by the set $\{(A_1, B_1), \ldots, (A_n, B_n)\}$. The dependency states that for every tuple t in the relation under consideration, there is a tuple s such that $t(A_1) = s(B_1)$ and $\ldots$ and $t(A_n) = s(B_n)$. In the following query, the binary schema-variable $\mathcal{X}^{(2)}$ is used to range over inclusion dependencies.

$$\{\mathcal{X}^{(2)} \mid \forall s(\exists t(\forall \mathcal{X}(Y, Z)(Y \neq Z \wedge s.Y = t.Z)))\} \quad (4)$$

For the relation:

R	A	B	C	D	E
	1	2	3	4	5
	1	1	2	1	5
	1	1	1	4	5
	1	2	5	1	5

the result is:

$\mathcal{X}^{(2)}$
$\{(A,B),(A,C)\}$
$\{(A,B),(A,D)\}$
$\{(A,B),(B,C)\}$
$\{(A,D),(E,C)\}$
$\{(A,B)\}$
$\{(A,C)\}$
$\{(A,D)\}$
$\{(B,C)\}$
$\{(E,C)\}$
$\{\}$

.

Note that the result is again subset-closed.

5 Subset-closed and Superset-closed DML Queries

The basic property underlying the a-priori algorithm [1] is that every subset of a frequent itemset is also a frequent itemset. This property is generalized for DML queries and called *subset-closed*:

Definition 1. *Let δ be a DML sentence with schema-variable $\mathcal{X}$. δ is* subset-closed in $\mathcal{X}$ *(or, $\mathcal{X}$-subset-closed) iff for every DML structure $\langle R, \Sigma\rangle$, for every $T \subseteq \Sigma(\mathcal{X})$, if $\langle R, \Sigma\rangle \models \delta$ then $\langle R, \Sigma_{\mathcal{X}\to T}\rangle \models \delta$. A DML sentence δ is* subset-closed *iff it is subset-closed in every schema-variable. A query $\{\mathcal{X}_1, \ldots, \mathcal{X}_m \mid \delta\}$ is* subset-closed *(in $\mathcal{X}_i$) iff δ is subset-closed (in $\mathcal{X}_i$, $i \in [1..m]$).*

Superset-closed *DML formulas and queries are defined in the same way (replace $T \subseteq \Sigma(\mathcal{X})$ by $T \supseteq \Sigma(\mathcal{X})$).*

Note incidentally that the construct of subset-closedness does not rely on a fixed underlying schema; that is, Definition 1 considers any relation R over any schema. Clearly, subset-closedness and superset-closedness are complementary notions, in the sense that the negation of a subset-closed DML sentence is superset-closed and *vice versa*.

Recognizing subset-closed queries is significant from a data mining perspective because these queries are amenable to query optimization by levelwise search, in the same way as the problem of mining frequent itemsets is solved by the a-priori algorithm. The search first examines which singletons are solutions, and then iteratively examines ever larger sets, but without examining any set that cannot be a solution because in earlier iterations, a proper subset of it turned out to be no solution. This is the general idea; it is worth pointing out that also on a more detailed level, techniques of the a-priori algorithm generalize to subset-closed queries, for example, the candidate generation consisting of join and prune steps [4, Chapter 6].

The same technique applies to superset-closed queries, in which case the search starts from the largest set and iteratively examines sets of lower cardinality, but without ever examining any set that cannot be a solution because one of its supersets was no solution.

If future optimizers for data mining queries have to incorporate a-priori optimization, then they should be able to recognize subset/superset-closed queries. Unfortunately, the class of subset-closed DML queries is not recursive.

Theorem 1. *Subset-closedness of DML queries is undecidable.*

Theorem 1 raises the problem of finding a recursive subclass of the class of subset-closed queries that semantically covers a large (or the entire) class of subset-closed queries. Positive DML queries are a candidate.

6 Positive DML Queries

Definition 2. *A DML formula δ that contains the schema-variable $\mathcal{X}$, is* positive in $\mathcal{X}$ *(or $\mathcal{X}$-positive) iff every symbol $\mathcal{X}$ lies within the scope of an even number of negations. A DML formula δ is* positive *iff it is positive in every schema-variable. A query $\{\mathcal{X}_1, \ldots, \mathcal{X}_m \mid \delta\}$ is* positive *(in $\mathcal{X}_i$) iff δ is positive (in $\mathcal{X}_i$, $i \in [1..m]$).*

Note incidentally that positive, unlike subset-closed, is defined for DML formulas that may not be closed.

Lemma 1. *Let δ be a DML sentence. If δ is $\mathcal{X}$-positive, then δ is $\mathcal{X}$-superset-closed. If $\neg\delta$ is $\mathcal{X}$-positive, then δ is $\mathcal{X}$-subset-closed.*

For example, the application of Lemma 1 tells us that the query (1) in Section 2 is subset-closed. By the same lemma , the query (2) for finding frequent sets and the query (4) for finding inclusion dependencies, both introduced in Section 4, are subset-closed. Note that abbreviations have to be spelled out before testing positiveness. In particular, $\forall\mathcal{X}(Y)(\delta)$ becomes $\forall Y(\neg\mathcal{X}(Y) \vee \delta)$. The lemma does not apply to the query (3) for finding functional dependencies. When the query is spelled out

$$\{\mathcal{X}, \mathcal{Y} \mid \forall t(\forall s((\exists X(\mathcal{X}(X) \wedge t.X \neq s.X)) \vee \neg(\exists Y(\mathcal{Y}(Y) \wedge t.Y \neq s.Y)))) \\ \wedge \exists Y(\mathcal{Y}(Y) \wedge \neg\mathcal{X}(Y))\} \ , \tag{5}$$

it turns out that both $\mathcal{X}$ and $\mathcal{Y}$ occur within an even and an odd number of negations.

Because subset-closed and superset-closed are complementary notions, it is sufficient to focus on superset-closed in what follows. Unfortunately, the (recursive) class of positive DML queries does not semantically cover the whole class of superset-closed DML queries; this negative result obtains even if only queries with a single schema-variable are considered.

7 A Superset-closed DML Query with a Single Schema-variable that Cannot be Expressed Positively

In Sections 8 and 9, we show that not every $\mathcal{X}$-superset-closed DML sentence is equivalent to some $\mathcal{X}$-positive DML sentence. The proof relies on Stolboushkin's

refutation [14] of Lyndon's Lemma (that every monotone first-order property is expressible positively) for finite models. Although Lyndon's Lemma was first refuted for finite models in [2], we rely on Stolboushkin's construction of a first order (FO) sentence Ω in the signature $\langle H, < \rangle$, where H and $<$ are two binary predicate symbols, such that Ω is finitely $<$-monotone,[5] but Ω is finitely equivalent to no $<$-positive FO sentence.

In general, the failure of Lyndon's Lemma for DML is not surprising, as DML is essentially FO in which one can talk about attributes; n-ary schema-variables correspond to n-ary predicate symbols, and attribute-variables to FO variables. The new contribution however is that Lyndon's Lemma fails for DML even if only DML queries with a single schema-variable $\mathcal{X}$ are considered. Such queries suffice for expressing several common data mining problems, like finding frequent sets or inclusion dependencies.

We define a mapping $\mathbf{D}(\cdot)$ from FO formulas to DML formulas and we show that $\mathbf{D}(\Omega)$ is (i) $\mathcal{X}$-superset-closed but (ii) equivalent to no $\mathcal{X}$-positive DML sentence. The mapping $\mathbf{D}(\cdot)$ is such that $<$-monotone FO sentences are mapped to $\mathcal{X}$-superset-closed DML sentences (Lemma 4), so that (i) is instantly verified. To establish (ii), we define a "backward" mapping $\mathbf{F}(\cdot)$ from DML formulas to FO formulas such that:

- $\mathcal{X}$-positive DML sentences are mapped through $\mathbf{F}(\cdot)$ on $<$-positive FO sentences (Lemma 2), and
- every DML sentence equivalent to $\mathbf{D}(\Omega)$ is mapped through $\mathbf{F}(\cdot)$ onto a FO sentence equivalent to Ω (Lemma 7).

In Sections 8 we define the mappings between DML and FO. Section 9 introduces the properties introduced above, which finally allow concluding the existence of an $\mathcal{X}$-superset-closed DML sentence that is equivalent to no $\mathcal{X}$-positive DML sentence (Theorem 2).

8 Mappings

8.1 Structure Encodings

The mapping $\mathcal{D}(\cdot)$ maps FO structures of signature $\langle H, < \rangle$ to DML structures. If I is a FO structure of signature $\langle H, < \rangle$, then $\mathcal{D}(I) = \langle R, \Sigma \rangle$ will "encode" the information in I using the following encoding scheme:

- The schema $|R|$ is the domain of I.
- Every tuple t of R *encodes* an element of $I(H)$ as follows. A tuple t with $t(A) = 1$, $t(B) = 2$, and $t(X) = 0$ otherwise, encodes that $(A, B) \in I(H)$. A tuple t with $t(A) = 3$ and $t(X) = 0$ otherwise, encodes that $(A, A) \in I(H)$. Every element in $I(H)$ is encoded in this way by a tuple in R; moreover, R contains no other tuples.

[5] Ω being finitely $<$-monotone means that for any (finite) structure I of signature $\langle H, < \rangle$, if $I \models \Omega$ and $I(<) \subseteq T$ than $I_{<\to T} \models \Omega$.

– A binary schema-variable $\mathcal{X}$ is used for encoding $<$, i.e., $\Sigma(\mathcal{X}) = I(<)$. No other schema-variables will occur in the mappings.

Example 1. Let I be a FO structure of signature $\langle H, < \rangle$ and universe $\{A, B, C, D, E\}$. Let $I(H) = \{(A,B), (A,D), (A,E), (C,E), (C,C)\}$. Then $\mathcal{D}(I)$ is a DML structure $\langle R, \Sigma \rangle$ with R as follows:

R	A	B	C	D	E
	1	2	0	0	0
	1	0	0	2	0
	1	0	0	0	2
	0	0	1	0	2
	0	0	3	0	0

8.2 Interpretation Encodings

The mapping $\mathcal{D}(\cdot)$ is extended from structures to interpretations. The DML formulas δ considered will contain attribute-variables, tuple-variables, and a single binary schema-variable $\mathcal{X}$. As explained before, the binary FO predicate symbol $<$ is tied to the schema-variable $\mathcal{X}$. In addition, FO variables are *tied* to simple-variables as follows:

– a FO variable x is tied to each attribute-variable X, and
– two FO variables t_1, t_2 are tied to each tuple-variable t.

Let I be a FO structure of signature $\langle H, < \rangle$, and ν a variable assignment to the variables tied to the simple-variables of the DML formula δ. The mapping $\mathcal{D}(\cdot)$ is extended to the *FO interpretation* $\langle I, \nu \rangle$ as follows; $\mathcal{D}(\langle I, \nu \rangle)$ is the DML interpretation $\langle \langle R, \Sigma \rangle, \sigma \rangle$ with the following characteristics:

– $\langle R, \Sigma \rangle = \mathcal{D}(I)$, as previously defined,
– $\sigma(X) = \nu(x)$ for every attribute-variable X, and
– for every tuple-variable t, if $(\nu(t_1), \nu(t_2)) \in I(H)$, then $\sigma(t)$ is the tuple in R encoding $(\nu(t_1), \nu(t_2))$ in $I(H)$; otherwise $\sigma(t)(X) = 4$ for all $X \in |R|$. Note that if $(\nu(t_1), \nu(t_2))$ does not belong to $I(H)$, then R contains no tuple encoding $(\nu(t_1), \nu(t_2))$. In that case $\sigma(t)$ is chosen to be a tuple that takes the value 4 in all attributes. Such tuple does not belong to R, as only the values 0, 1, 2, and 3 can occur in R under the given encoding.

8.3 Formula Mappings

The formula mappings are coined with the mapping $\mathcal{D}(\cdot)$ from FO interpretations to DML interpretations in mind. The formula mappings $\mathbf{D}(\cdot)$ and $\mathbf{F}(\cdot)$ are established such that $I \models \phi$ if and only if $\mathcal{D}(I) \models \mathbf{D}(\phi)$ (Lemma 3), and

$\mathcal{D}(I) \models \delta$ if and only if $I \models \mathbf{F}(\delta)$ (Lemma 5). The mapping $\mathbf{D}(\cdot)$ maps FO formulas of signature $\langle H, < \rangle$ to DML formulas and is defined next:

$$
\begin{aligned}
C_1 &= \forall t(\forall X(t.X = 0 \vee t.X = 1 \vee t.X = 2 \vee t.X = 3)) \\
C_2 &= \forall t(\exists X(t.X = 1 \vee t.X = 2 \vee t.X = 3)) \\
C_3 &= \forall t(\forall X(\forall Y((X \neq Y \wedge t.X = t.Y) \rightarrow t.X = 0))) \\
C_4 &= \forall t((\exists X(t.X = 1)) \leftrightarrow (\exists Y(t.Y = 2))) \\
C_5 &= \forall t((\exists X(t.X = 3)) \rightarrow \neg\exists Y(t.Y = 1 \vee t.Y = 2))) \\
C &= (C_1 \wedge \ldots \wedge C_5)
\end{aligned}
$$

The DML formula C is such that every DML structure satisfying C can be thought of as the "encoding" by $\mathcal{D}(\cdot)$ of some FO structure I of signature $\langle H, < \rangle$.

$$
\begin{aligned}
\mathbf{D}(\phi) &= \mathbf{D}'(\phi) \wedge C \\
\mathbf{D}'(\neg\phi) &= \neg\mathbf{D}'(\phi) \\
\mathbf{D}'(\phi_1 \vee \phi_2) &= \mathbf{D}'(\phi_1) \vee \mathbf{D}'(\phi_2) \\
\mathbf{D}'(\exists x(\phi)) &= \exists X(\mathbf{D}'(\phi)) \\
\mathbf{D}'(H(x, y)) &= \exists t((t.X = 1 \wedge t.Y = 2) \vee (t.X = 3 \wedge t.Y = 3)) \\
\mathbf{D}'(x < y) &= \mathcal{X}(X, Y)
\end{aligned}
$$

The mapping $\mathbf{F}(\cdot)$ maps DML formulas to FO formulas.

$$
\begin{aligned}
\mathbf{F}(X = Y) &= x = y \\
\mathbf{F}(t.X = s.Y) &= \mathbf{F}((t.X = 0 \wedge s.Y = 0) \vee \ldots \vee (t.X = 4 \wedge s.Y = 4)) \\
\mathbf{F}(t.X = 0) &= x \neq t_1 \wedge x \neq t_2 \wedge H(t_1, t_2) \\
\mathbf{F}(t.X = 1) &= x = t_1 \wedge x \neq t_2 \wedge H(t_1, t_2) \\
\mathbf{F}(t.X = 2) &= x \neq t_1 \wedge x = t_2 \wedge H(t_1, t_2) \\
\mathbf{F}(t.X = 3) &= x = t_1 \wedge x = t_2 \wedge H(t_1, t_2) \\
\mathbf{F}(t.X = 4) &= \neg H(t_1, t_2) \\
\mathbf{F}(t.X = a) &= \textbf{false} \text{ if } a \notin \{0, 1, 2, 3, 4\} \\
\mathbf{F}(\mathcal{X}(X, Y)) &= x < y \\
\mathbf{F}(\neg\delta) &= \neg\mathbf{F}(\delta) \\
\mathbf{F}(\delta_1 \vee \delta_2) &= \mathbf{F}(\delta_1) \vee \mathbf{F}(\delta_2) \\
\mathbf{F}(\exists X(\delta)) &= \exists x(\mathbf{F}(\delta)) \\
\mathbf{F}(\exists t(\delta)) &= \exists t_1(\exists t_2(H(t_1, t_2) \wedge \mathbf{F}(\delta)))
\end{aligned}
$$

In what follows, δ denotes a DML formula, and ϕ a FO formula of signature $\langle H, < \rangle$. I denotes a FO structure of signature $\langle H, < \rangle$. $\langle R, \Sigma \rangle$ denotes a DML structure; since $\mathcal{X}$ is the only schema-variable involved, it suffices to specify $\Sigma(\mathcal{X})$.

9 A-priori and Finitely Monotone Properties

The mappings defined in the previous section satisfy certain properties which eventually allow concluding that the class of positive queries does not semantically cover the whole class of superset-closed queries.

Lemma 2. *Let δ be a DML formula. If δ is $\mathcal{X}$-positive, then $\mathbf{F}(\delta)$ is $<$-positive.*

We recall that detailed proofs of all lemmas and theorems can be found in [3].

Lemma 3. *Let ϕ be a FO sentence of signature $\langle H, < \rangle$. $I \models \phi$ iff $\mathcal{D}(I) \models \mathbf{D}(\phi)$.*

Corollary 1. *Let ϕ_1 and ϕ_2 be two FO sentences of signature $\langle H, < \rangle$. If $\mathbf{D}(\phi_1) \equiv \mathbf{D}(\phi_2)$, then $\phi_1 \equiv \phi_2$.*

Lemma 4. *Let ϕ be a FO sentence of signature $\langle H, < \rangle$. If ϕ is $<$-monotone, then $\mathbf{D}(\phi)$ is $\mathcal{X}$-superset-closed.*

Lemma 5. *Let δ be a DML sentence. $I \models \mathbf{F}(\delta)$ iff $\mathcal{D}(I) \models \delta$.*

Lemma 6. *Let δ be a DML sentence. If $\delta \models C$, then $\delta \equiv \mathbf{D}(\mathbf{F}(\delta))$.*

Lemma 7. *Let δ be a DML sentence and ϕ a FO sentence of signature $\langle H, < \rangle$. If $\mathbf{D}(\phi) \equiv \delta$, then $\phi \equiv \mathbf{F}(\delta)$.*

Theorem 2. *There exists an $\mathcal{X}$-superset-closed DML sentence with a single schema-variable $\mathcal{X}$ that is equivalent to no $\mathcal{X}$-positive DML sentence.*

Proof. By [14], we can assume the existence of a FO sentence Ω of signature $\langle H, < \rangle$ that is finitely $<$-monotone but that is finitely equivalent to no $<$-positive FO sentence. By Lemma 4, $\mathbf{D}(\Omega)$ is $\mathcal{X}$-superset-closed. It suffices to show that $\mathbf{D}(\Omega)$ is equivalent to no $\mathcal{X}$-positive DML sentence. Assume on the contrary the existence of an $\mathcal{X}$-positive DML sentence δ that is equivalent to $\mathbf{D}(\Omega)$. By Lemma 7, $\Omega \equiv \mathbf{F}(\delta)$. Since δ is $\mathcal{X}$-positive, $\mathbf{F}(\delta)$ is $<$-positive (Lemma 2). This contradicts our assumption about Ω. □

10 Discussion and Future Work

DML allows expressing data mining queries that ask for rules involving sets of attributes, like "Find frequent itemsets" and "Find functional dependencies." The a-priori technique directly applies to all subset-closed and superset-closed DML queries. Since a DML query is subset-closed if and only if its negation is superset-closed, it suffices to focus on superset-closedness. Superset-closedness of DML queries is undecidable. It is an open problem whether there exists a recursive subclass of superset-closed DML queries such that every superset-closed DML query is equivalent to some query in this subclass. We revealed a significant relationship between superset-closedness in DML and finitely monotone first order properties. Each positive DML query is superset-closed, but there exist

superset-closed queries with a single schema-variable that cannot be expressed positively. One may hope that the class of all positive DML queries and their negations semantically covers all "practical" DML queries that are amenable to a-priori optimization. Anyway, as long as we do not know a recursive subclass of the class of superset-closed DML queries that semantically covers the whole class of superset-closed DML queries, query optimizers can use positiveness of a query as a criterion for determining the applicability of a-priori pruning.

We list two interesting problems for future work. The first problem concerns the extension of DML with aggregate functions, which are obviously needed in many data mining tasks. The query (2) for finding frequent sets, introduced in Section 4, can probably be expressed more naturally by using some *count* function. An interesting problem is to find recursive subclasses of subset/superset-closed queries in the presence of aggregate functions.

Secondly, certain subqueries of a DML query can be subset/superset-closed, even though the query itself is not. In this case, a-priori optimization could still be applied on these subqueries. Recall, for example, that the DML query for finding non-trivial functional dependencies (queries (3) and (5) in Sections 4 and 6 respectively) is neither subset-closed nor superset-closed. However, if we omit the requirement that the functional dependencies be non-trivial, the query (5) reduces to:

$$\{\mathcal{X}, \mathcal{Y} \mid \forall t(\forall s((\exists X(\mathcal{X}(X) \wedge t.X \neq s.X)) \vee \neg(\exists Y(\mathcal{Y}(Y) \wedge t.Y \neq s.Y))))\} \ . \quad (6)$$

By Lemma 1, the query (6) is $\mathcal{X}$-superset-closed and $\mathcal{Y}$-subset-closed, which is tantamount to saying that if the functional dependency $X \rightarrow Y$ is satisfied, and $X \subseteq X'$ and $Y' \subseteq Y$, then $X' \rightarrow Y'$ must necessarily be satisfied as well. This property again allows a-priori pruning in the following way: In order to find all non-trivial functional dependencies satisfied by a given relation (query (5)), one could use the a-priori trick on the query that finds all functional dependencies that are satisfied (query (6)), and filter away those that are trivial. The filter corresponds to the following subquery of query (5) that was left out in query (6):

$$\{\mathcal{X}, \mathcal{Y} \mid \exists Y(\mathcal{Y}(Y) \wedge \neg\mathcal{X}(Y))\} \ . \quad (7)$$

That the latter query is $\mathcal{X}$-subset-closed and $\mathcal{Y}$-superset-closed, is of second importance. This way of optimizing queries by exploiting the a-priori technique on subset/superset-closed subqueries, resembles the idea underlying query flocks [15]; we believe that further research is needed to assess its feasibility and usefulness.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., 1993.
2. M. Ajtai and Y. Gurevich. Monotone versus positive. *Journal of the ACM*, 34(4):1004–1015, 1987.

3. T. Calders and J. Wijsen. On monotone data mining languages. Technical Report 2001-08, Universitaire Instelling Antwerpen, Department of Mathematics & Computer Science, 2001.
4. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.
5. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Comm. of the ACM*, 39(11):58–64, 1996.
6. M. Kantola, H. Mannila, K.-J. Räihä, and H. Siirtola. Discovering functional and inclusion dependencies in relational databases. *Internat. Journal of Intelligent Systems*, 7:591–607, 1992.
7. L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 157–168, 1999.
8. L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL—An extension to SQL for multi-database interoperability. *To appear in ACM Trans. on Database Systems*.
9. S. Lopes, J.-M. Petit, and L. Lakhal. Efficient discovery of functional dependencies and Armstrong relations. In *Proc. 7th Int. Conf. on Extending Database Technology (EDBT 2000)*, LNCS 1777, pages 350–364. Springer, 2000.
10. H. Mannila. Methods and problems in data mining. In *Proc. Int. Conf. on Database Theory*, Delphi, Greece, 1997.
11. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
12. R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 13–24. ACM Press, 1998.
13. K. A. Ross. Relations with relation names as arguments: Algebra and calculus. In *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 346–353. ACM Press, 1992.
14. A. Stolboushkin. Finitely monotone properties. In *Proc. 10th IEEE Symp. on Logic in Comp. Sci.*, pages 324–330, 1995.
15. D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: a generalization of association-rule mining. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 1–12, 1998.
16. J. Wijsen, R. Ng, and T. Calders. Discovering roll-up dependencies. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pages 213–222, San Diego, CA, 1999.