ON THE EXPRESSIVE POWER OF DATA DEPENDENCIES

by

J.A. Makowsky* and M.Y. Vardi**

TECHNICAL REPORT No. 326

July 1984

* Dept. of Computer Science, Technion - IIT, Haifa, Israel
** IBM Research Laboratory, San Jose, CA 95193
ON THE EXPRESSIVE POWER OF DATA DEPENDENCIES

Johann A. Makowsky
Dept. of Computer Science
Technion - Israel Institute of Technology
Haifa, Israel

Moshe Y. Vardi
IBM Research Laboratory
San Jose, CA 95193

Abstract

The class of data dependencies is a class of first-order sentences that seem to be suitable to express semantic constraints for relational databases. We deal with the question of which classes of databases are axiomatizable by data dependencies. (A class $\Gamma$ of databases is said to be axiomatizable by sentences of a certain kind if there exists a set of sentences of that kind such that $\Gamma$ is the class of all models of that set.) Our results characterize, by algebraic closure conditions, classes of databases that are axiomatizable by dependencies of different kinds. Our technique is model-theoretic, and the characterization easily entails all previously known results on axiomatizability by dependencies.

* Research partially supported by Swiss National Science Foundation Grant No. 82.820.0.80
† The research reported here was done while the author was at Stanford University and supported by a Weizmann Post-Doctoral Fellowship and AFOSR grant 80-0212.
1. Introduction

The class of *data dependencies* is a class of first-order sentences of a specific syntax, which we will describe later, that seem to be suitable to express semantic constraints for relational databases. The study of dependencies began with the *functional dependencies* of [Co2]. After the introduction of *multiple valued dependencies* by [Fag1, Zan] the field became chaotic for a few years in which researchers introduced many new classes of dependencies. Recently, two unifying formalisms have been suggested and turned out to be equivalent. The class of *embedded implicational dependencies* [BV, Fag2], which is equivalent to the class of *algebraic dependencies* [YP], seems to contain most cases of interest. Intuitively, a dependency is a first-order sentence that says that if some tuples, fulfilling certain equalities, exist in the database then either some other tuples must also exist in the database or some values in the given tuples must be equal.

In trying to understand the expressive power of data dependencies, two lines of investigation have been taken. The first line of investigation, initiated in [Fag2] and pursued in [CLM, Mak1] and, based on an early draft of this paper in [Mak2], tries to characterize dependencies by preservation theorems. The following is a typical such characterization: "a first-order sentence is equivalent to a conjunction of embedded implicational dependencies if and only if it is domain independent, holds in trivial relations, and is preserved under union of chains and direct product" [CLM]. We call this approach the *preservation approach*, since it focuses on the preservation properties of dependencies.

We find the preservation approach unsatisfying from two points of view. First, the given characterizations usually use compactness, so one is required to talk about infinite databases, which is quite disturbing. However, very often the compactness argument can be replaced by other proof methods, which also apply to finite model theory, see e.g. [Gu] and [Mak3]. Secondly, while that approach has lead to a better understanding of the preservation properties of data dependencies, it does not lead immediately to a better understanding of the expressive power of data dependencies. That is, it does not deal with the question of which classes of databases are *axiomatizable* by data dependencies. (A class $\Gamma$ of databases is said to be axiomatizable by sentences of a certain kind if there exists a set of sentences of that kind such that $\Gamma$ is the class of all models of that set.) One should note, though, that, using compactness, one can usually transform a preservation theorem into an axiomatizability theorem. To obtain axiomatizability results in finite model theory one again has to search for modified proofs.

This line of investigation, which we call the *axiomatizability approach*, since it focuses on properties of classes of databases, was initiated in [GZ] and pursued in [Fag2, GH, Hu, GV]. All these works dealt with the following question. Let $\Gamma$ be a class of databases axiomatizable by dependencies of a certain kind, and let $\Delta$ be the class of databases obtained by applying a certain operation $\alpha$ to $\Gamma$. Is $\Delta$ axiomatizable by dependencies of that kind? The practical problem that inspired this question is the implementation of external user views, where $\Gamma$ is a class of conceptual databases and $\Delta$ is a class of external user views defined by a view definition $\alpha$ [JAK]. Typical axiomatizability results are "The projection of a class of databases axiomatizable by functional dependencies is not
necessarily axiomatizable by functional dependencies" [GZ], or "the projection of a class of databases axiomatizable by implicational dependencies is axiomatizable by implicational dependencies" [Fag2].

Even though the results in the aforementioned works on axiomatizability approach do give us a somewhat better understanding of the picture, they are still a sequence of ad-hoc results, and do not supply us with any comprehensive theory. Thus, for every possible operation \( \alpha \) that defines \( \Delta \) via \( \Gamma \), one has essentially to answer the question from scratch. Also, these works deal only with databases that consist of a single relation.

Our goal in this paper is to supply this missing comprehensive theory. We do this by characterizing axiomatizability algebraically in the style of Birkhoff [Bi] and Mal'cev [Mal], cf. also [Mak2]. That is, our theorems will characterize, by algebraic closure conditions, classes of databases that are axiomatizable by dependencies of different kinds. The following is a typical theorem: "A class of databases is axiomatizable by full dependencies if, and only if it contains a trivial database, is domain independent, and is closed under subdatabases and direct products." These characterizations easily entail the aforementioned axiomatizability results as well as stronger results. For example, it is easy to show that the projection of a class of databases axiomatizable by implicational dependencies contains a trivial database, is domain independent, and is closed under subdatabases and direct products.

To prove our characterization we use two classical model-theoretic techniques: the method of diagrams [CK] and McKinsey's method [McK]. This enables us to prove results about embedded dependencies (e.g., Theorem 7) for which the proof-theoretic technique of [Fag2] does not seem to be applicable.

2. Dependencies

In the relational model [Co1], the database is viewed as a finite relational structure \( B=(A,R_1,\ldots,R_m) \), consisting of a finite domain \( A \) and relations \( R_1,\ldots,R_m \) over that domain. For any given application only a subclass of all possible databases is of interest. This subclass is defined by semantic constraints that are to be satisfied by the databases of interest. A family of constraints that was extensively studied in the literature is the family of dependencies. (The reader who is interested in the relationship between the family of dependencies defined here and other families of dependencies is referred to [Fag2].)

Let the language \( L \) consist of relation names \( R_1,\ldots,R_m \), whose intended interpretations are the relations \( R_1,\ldots,R_m \) of the database. We denote the arity of \( R_i \) by \( n_i \). We call an atomic formula of the form \( R_i(v_1,\ldots,v_{n_i}) \) a relational formula, and an atomic formulas of the form \( v_p = v_q \) an equality formula. We use \( x \)'s and \( y \)'s as variables names, and \( v \)'s and \( u \)'s as meta-variables that range over variables names. A dependency is a first-order sentence in the language \( L \) of the form
where:

1. \( k, p, q \geq 1 \) and \( l \geq 0 \).
2. The \( A \)'s are relational formulas that use between themselves exactly all the variables \( y_1, \ldots, y_k \).
3. The \( B \)'s use between themselves all the variables \( x_1, \ldots, x_l \) and possibly some \( y \)'s.
4. Either all the \( B \)'s are relational formulas or \( l = 0 \) and they are all equality formulas.

If all \( B \)'s are relational formulas, the dependency is called a \textit{tuple generating dependency} (abbr. tgd). Intuitively, a tgd says that if some tuples, satisfying certain equalities, exist in the database, then some other tuples, satisfying certain other equalities, must also exist in the database. If all the \( B \)'s are equality formulas, the dependency is called an \textit{equality generating dependency} (abbr. egd). Intuitively, an egd says that if some tuples, satisfying certain equalities, exist in the database, then these tuples must also satisfy some other certain equalities.

Dependencies without existential quantifiers, i.e., in the syntax above \( l = 0 \), are called \textit{full}. Observe that egd's are necessarily full. Observe also that every full dependency is equivalent to a conjunction of finitely many full dependencies with \( q = 1 \), i.e., there is a single atomic formula on the right-hand-side of the implication. Thus, we assume, without loss of generality that all full dependencies are of this form. We say that a full dependency

\[
\forall y_1 \cdots y_k (A_1 \wedge \cdots \wedge A_p \rightarrow B)
\]

has \( k \) variables.

A \textit{typing} for the language \( L \) is a pair \((I, \alpha)\), consisting of an index set \( I \) and a mapping \( \alpha \) from the set \( \{<i,j>: 1 \leq i \leq n_1, 1 \leq j \leq n_r \} \) onto \( I \), such that \( \alpha(i,j) = \alpha(i,k) \) entails \( j = k \). Intuitively, \( I \) is a set of \textit{sorts}, and \( \alpha \) is an assignment of sorts to the columns of the relations such that no two columns of a relation have the same sort. The motivation is that though an employee's number and his salary may happen to be equal, they have really nothing to do with each other. Let now \( D \) be a set of dependencies. We say that \( D \) is \((I, \alpha)-\text{typed}\) when, for all \( \sigma \) in \( D \), if both \( R_i(u_1, \ldots, u_r, \ldots, y_{n_1}) \) and \( R_j(u_1, \ldots, u_s, \ldots, u_{n_2}) \) occur in \( \sigma \) and either \( u_r \) and \( u_s \) are identical or \( u_r = u_s \) occurs in \( \sigma \), then \( \alpha(i,r) = \alpha(j,s) \). Intuitively, \( D \) is \((I, \alpha)-\text{typed}\) if it is "compatible" with the typing \((I, \alpha)\). For example, the dependency:

\[
\forall y_1 y_2 y_3 (R_1(y_1, y_2) \wedge R_2(y_1, y_3) \rightarrow y_2 = y_3)
\]

is \((I, \alpha)-\text{typed}\), for \( I = \{1, 2\} \), \( \alpha(1,1) = 1 \), and \( \alpha(1,2) = 2 \). The set consisting of the dependencies \( \forall y_1 y_2 (R_1(y_1, y_2) \rightarrow R_3(y_1, y_2)) \) and \( \forall y_1 y_2 (R_1(y_1, y_2) \rightarrow R_2(y_2, y_1)) \) is not \((I, \alpha)-\text{typed}\) for any typing \((I, \alpha)\).

We note that dependencies for a special class of universal existential Horn formulas of first order logic without function symbols. For the special significance of Horn formulas in data base
3. Preservation Properties

In order to characterize axiomatizability we translate preservation properties into closure properties. We first review some known preservation properties of dependencies, and describe some new ones. We refer in this section to databases $B=(A_1,R_1,\ldots,R_m)$, $B_1=(A_1,R_1,\ldots,R_m)$, and $B_2=(A_2,R'_1,\ldots,R'_m)$.

$B$ is trivial if $|A|=1$ and $|R_i|=1$, for $1 \leq i \leq m$.

Lemma 1. [CLM] Dependencies are satisfied in all trivial databases.

$B_1$ and $B_2$ are said to be similar, if they have exactly the same relations, that is $R_1'=R'_2$, for $1 \leq i \leq m$. A dependency $\sigma$ is said to be domain independent if for all similar databases $B_1$ and $B_2$ we have that $B_1$ satisfies $\sigma$ if and only if $B_2$ satisfies $\sigma$.

Lemma 2. [Ku,Fag2] Dependencies are domain independent.

We say that $B_2$ is contained in $B_1$ if $A_2 \subseteq A_1$ and $R_2^i \subseteq R_1^i$, for $1 \leq i \leq m$. We say that $B_2$ is a subdatabase of $B_1$ if $A_2 \subseteq A_1$ and $R_2^i$ is the restriction of $R_1^i$ to $A_2$, for $1 \leq i \leq m$. A dependency $\sigma$ is said to be preserved under containment if whenever $B_1$ satisfies $\sigma$ and $B_2$ is contained in $B_1$, then $B_2$ also satisfies $\sigma$. Preservation under subdatabases is defined analogously.

Lemma 3. Full dependencies are preserved under subdatabases [CLM]. Equids are also preserved under containment.

We now consider closure in the other direction (from subsets to supersets). If $|A|=n$, we say that $B$ has $n$ elements.

Lemma 4. Let $\sigma$ be a full dependency with $n$ variables. If $\sigma$ is satisfied by all databases $B_2$ that are subdatabases of $B_1$ and have at most $n$ elements, then $\sigma$ is satisfied by $B_1$.

We now define the direct product of $B_1$ and $B_2$, denoted $B_1 \otimes B_2$. It is the database

$$(A_1 \times A_2, R_1^1 \otimes R_2^1, \ldots, R_m^1 \otimes R_m^2),$$

where $A_1 \times A_2$ is the Cartesian product of $A_1$ and $A_2$, and $R_1^1 \otimes R_2^2$ is the relation

$$\{\langle a_1, a_2^1, a_2^2, \ldots, a_n^1, a_n^2 \rangle : \langle a_1^1, \ldots, a_n^1 \rangle \in R_1^1 \text{ and } \langle a_1^2, \ldots, a_n^2 \rangle \in R_2^2\}.$$ 

A dependency $\sigma$ is said to be preserved under direct products if whenever it is satisfied by $B_1$ and $B_2$, then it is also satisfied by $B_1 \otimes B_2$.

Lemma 5. [Fag2] Dependencies are preserved under direct products.

We now define the intersection of $B_1$ and $B_2$, denoted $B_1 \cap B_2$. It is the database

$$(A_1 \cap A_2, R_1^1 \cap R_2^1, \ldots, R_m^1 \cap R_m^2).$$

theory see also [Fä2, CH, Mak2, Mak3]. Horn formulas also provide for the syntactic framework of logic programming. In [Mak3] the question is investigated as to why Horn formulas seem to play a central role in computer science.
A dependency $\sigma$ is said to be **preserved under intersections** if whenever it is satisfied by $B_1$ and $B_2$, then it is also satisfied by $B_1 \cap B_2$.

**Lemma 6.** Full dependencies are preserved under intersections.

Let $a \in A$ and let $b \in A$. Let $h$ be a mapping such that it is the identity on $A - \{a\}$ and $h(a) = b$. Extend $h$ to relations in the natural way. The database

$$B'(A \cup \{b\}, R_1 \cup h(R_1), \ldots, R_m \cup h(R_m))$$

is called a **duplicating extension** of $B$. Intuitively, $B'$ is obtained by putting into $B$ an extra copy of $a$. $\sigma$ is said to be preserved under **duplicating extensions** if whenever it is satisfied by $B_1$ and $B_2$ is a duplicating extension of $B_1$, then it is also satisfied by $B_2$.

**Lemma 7.** Tgd's are preserved under duplicating extensions.

Let $(f, \alpha)$ be a typing for $L$. An **$(f, \alpha)$-renaming** for $B$ is an $I$-indexed family $h_I = \{h_i : i \in I\}$ of automorphisms on $A$. $h_I(R_j)$ is the relation

$$\{h_{a(i, 1)}(a_1), \ldots, h_{a(i, n)}(a_n) : <a_1, \ldots, a_n> \in R_j\}.$$  

$h_I(B)$ is the database

$$\left(\bigcup_{i \in I} h_I(A), h_I(R_1), \ldots, h_I(R_m)\right).$$

An $(f, \alpha)$-renaming essentially treats occurrences of elements in columns of different sorts as distinct elements. A dependency $\sigma$ is said to be **preserved under $(f, \alpha)$-renamings**, if whenever $B$ satisfies $\sigma$ and $h_I$ is an $(f, \alpha)$-renaming, then $h_I(B)$ also satisfies $\sigma$.

**Lemma 8.** Let $(f, \alpha)$ be a typing for $L$, and let $\sigma$ be an $(f, \alpha)$-typed dependency. Then $\sigma$ is preserved under $(f, \alpha)$-renamings.

### 4. Dependency preserving insertions

From the preservation properties in the previous section only the preservation under cartesian products seems not to be relevant to database manipulation occurring in practice. In this section we introduce an update operation which seems to have a natural meaning and allows us to replace preservation under cartesian products in all the later theorems.

Let $\Sigma$ be a set of dependencies and $B = (A, R_1, \ldots, R_m)$ be a relational structure satisfying $\Sigma$. Let further $B = (b_1, \ldots, b_k)$ be a $k$-tuple of elements possibly not in $A$. Assume w.l.o.g. that $R_1$ is $k$-ary and that we want to add the new tuple $b$ to $R_1$ obtaining in this way the structure

$$B_1 = (A \cup \{b_1, \ldots, b_k\}, R_1 \cup \{b\}, R_2, \ldots, R_m).$$

Clearly, in general, $B_1$ will not satisfy $\Sigma$. The best we can hope for, is to find an extension $B^*$ of $B_1$ satisfying $\Sigma$ which is minimal, i.e. for every $B'$ containing $B_1$ we have that also $B^*$ is contained in $B'$.  

---

Technion - Computer Science Department - Technical Report CS0326 - 1984
We say that a set \( \Sigma \) of dependencies allows \textbf{unique minimal insertion updates}, if every for relational structure \( B \), satisfying \( \Sigma \) and every set of new tuples added to the relations there is a unique minimal relational structure \( B^* \) satisfying \( \Sigma \).

\textbf{Lemma 9.} \cite{Mak3} Full-dependencies allow unique minimal insertion updates.

\textbf{Lemma 10.} \cite{Mak3} If \( \Sigma \) is set of dependencies which allows unique minimal insertion updates then \( \Sigma \) is preserved under direct products and intersections.

In \cite{Mak3} dependencies allowing unique minimal insertion updates are completely characterized. Roughly speaking, they are exactly those dependencies where the existential quantifiers are replaced by the quantifier "there exists exactly one tuple \( \exists \) such that ...". For the exact formulation and the proof the reader is referred to \cite{Mak3}.

5. \textbf{Axiomatizability}

As noted above, we view dependencies as a language for semantic specification. In the database design process the designer decides upon the class of databases that are meaningful for the application in hand. (We assume that the only classes of databases considered are those closed under isomorphism.) Now the designer writes down a set \( \Sigma \) of dependencies such that the class of meaningful databases is the class of databases that satisfy \( \Sigma \), i.e., the class of finite models of \( \Sigma \), denoted \( SAT(\Sigma) \).

Let \( \Theta \) be a kind of dependencies (e.g., typed equality-generating); and let \( \Gamma \) be a class of databases. We say that \( \Gamma \) is \textbf{axiomatizable} by \( \Theta \)-dependencies if there exists a set \( D \) of \( \Theta \)-dependencies such that \( \Gamma = SAT(D) \). If \( D \) is a finite set, then we say that \( \Gamma \) is \textbf{finitely axiomatizable} by \( \Theta \)-dependencies. We are interested here in characterizing axiomatizability and finite axiomatizability by the families of full dependencies, full tgd's, and egd's.

Let us now define closure properties analogous to the preservation properties of the previous section. Let \( \Gamma \) be a class of databases. \( \Gamma \) is said to be \textbf{domain independent} if for all similar databases \( B_1 \) and \( B_2 \) we have that \( B_1 \) is in \( \Gamma \) if and only if \( B_2 \) is in \( \Gamma \). \( \Gamma \) is said to be \textbf{closed under containment} if whenever \( B_1 \) is in \( \Gamma \) and \( B_2 \) is contained in \( B_1 \), then \( B_2 \) is also in \( \Gamma \). We define closure under subdatabases, direct products, intersections, and duplicating extensions, analogously. We say that \( \Gamma \) is \textbf{closed under insertion updates} if whenever \( B \) is in \( \Gamma \) and we add a set of new tuples to the relations of \( B \) then there is a unique minimal \( B^* \) in \( \Gamma \) containing these new tuples.

We are now in position to characterize axiomatizability. The only if parts in the following theorems follow from the preservation properties of the previous section.

\textbf{Theorem 1.} A class \( \Gamma \) of databases is axiomatizable by full dependencies if and only if it contains a trivial database and it is domain independent and closed under subdatabases and direct products.

\textbf{Outline of Proof.} First, using the fact that \( \Gamma \) contains a trivial model and is domain independent and closed under subdatabases, we prove by the \textit{method of diagrams} \cite{CK} that \( \Gamma \) is axiomatizable by sentences of the form...
where $k, p, q \geq 1$, and the $A$'s are relational formulas that use between themselves exactly all the variables $y_1, \ldots, y_k$. Now, using the fact that $\Gamma$ is closed under direct products, we prove by McKinsey's method [McK] that $q \leq 1$. It follows that $\Gamma$ is axiomatizable by full dependencies.

Using lemma 10 we immediately obtain:

**Corollary.** A class $\Gamma$ of databases is axiomatizable by full dependencies if and only if it contains a trivial database and it is domain independent and closed under subdatabases and insertion updates.

The following theorems are proved analogously to the proof of Theorem 1, while we use the added closure conditions to refine the method of diagrams.

**Theorem 2.** A class $\Gamma$ of databases is axiomatizable by egd's if and only if it contains a trivial database and it is domain independent and closed under containment and direct products.

**Corollary.** A class $\Gamma$ of databases is axiomatizable by egd's if and only if it contains a trivial database and it is domain independent and closed under containment and insertion updates.

**Theorem 3.** A class $\Gamma$ of databases is axiomatizable by full tgd's if and only if it contains a trivial database and it is domain independent and closed under duplicating extensions and intersections.

Theorems 1, 2, and 3 can be relativized to a specific typing for $L$. We do it here only for full dependencies, and leave the cases of egd's and full tgd's to the reader. A class $\Gamma$ of databases is said to be closed under $(I, \alpha)$-renamings, if whenever $B$ is in $\Gamma$ and $h_I$ is an $(I, \alpha)$-renaming, then $h_I(B)$ is also in $\Gamma$.

**Theorem 4.** Let $(I, \alpha)$ be a typing for $L$. A class $\Gamma$ of databases is axiomatizable by $(I, \alpha)$-typed full dependencies if and only if it contains a trivial database and it is domain independent and closed under subdatabases, direct products, and $(I, \alpha)$-renamings.

**Corollary.** Let $(I, \alpha)$ be a typing for $L$. A class $\Gamma$ of databases is axiomatizable by $(I, \alpha)$-typed full dependencies if and only if it contains a trivial database and it is domain independent and closed under subdatabases, insertion updates, and $(I, \alpha)$-renamings.

A closure property analogous to the preservation property of Lemma 4 is now used to characterize finite axiomatizability. Again we do it here only for full dependencies. A class $\Gamma$ of databases is $n$-local if whenever $\Gamma$ contains all databases $B_2$ that are subdatabases of $B_1$ and have at most $n$-elements, then $\Gamma$ contains also $B_1$. $\Gamma$ is local if it is $n$-local for some $n$.

**Theorem 5.** A class $\Gamma$ of databases is finitely axiomatizable by full dependencies if and only if it contains a trivial database and it is local, domain independent, and closed under subdatabases and direct products.

**Corollary.** A class $\Gamma$ of databases is finitely axiomatizable by full dependencies if and only if it contains a trivial database and it is local, domain independent, and closed under subdatabases and
insertion updates.

In [Mak4] \(n\)-securable quantifiers are studied. There the semantics of a generalized quantifier is given by a class of structures \(\Gamma\). exactly as in the context here. It turns out the a class is \(n\)-securable iff it is \(n\)-local. Our notion of locality is also similar to the notion of boundedness in [GV]. There the bound is on the number of tuples in the database instead of the number of elements. Our notion is also reminiscent of the subinstance property of [Hu], but is simpler. (In our terminology, a class \(\Gamma\) of databases has the subinstance property for \(\mathcal{K}\), if the following holds: if every database \(B_3\) that is contained in \(B_1\) and has at most \(n\) tuples is also contained in a database \(B_3 \in \Gamma\) such \(B_3\) is contained in \(R_1\), then \(B_1 \in \Gamma\).)

6. Applications

In this section we apply the characterizations of the previous section to the issue of axiomatizability for external views and consistent databases. Since we have to use here several distinct language, we mention explicitly the relation names used in the language. In particular, \(L(P)\) is a language that consist of the relation name \(P\). Recall that the language of the database is \(L(R_1, \ldots, R_m)\).

Our first application is the axiomatizability of external views: The idea is that certain users need not see the database as it is but rather an image of it under some operation. A view definition is a formula \(\varphi(x_1, \ldots, x_n)\) in the language \(L(R_1, \ldots, R_m)\) with free variables \(x_1, \ldots, x_n\). Let \((I, \alpha)\) be a typing for \(L(R_1, \ldots, R_m)\). \(\varphi\) is \((I, \alpha)\)-typed if whenever \(R_i(u_1, \ldots, v_r, \ldots, u_m)\) and \(R_j(u_1, \ldots, u_s, \ldots, u_{r_j})\) occur in \(\varphi\) and either \(v_r\) and \(u_s\) are identical or \(v_r = u_s\) occurs in \(\varphi\), then \(\alpha(i, r) = \alpha(j, s)\), and whenever \(R_i(u_1, \ldots, v_r, \ldots, u_m)\) and \(R_j(u_1, \ldots, u_s, \ldots, u_{r_j})\) occur in \(\varphi\) and \(v_r\) and \(u_s\) are distinct free variables, then \(\alpha(i, r) \neq \alpha(j, s)\). We say that the type of \(v_r\) is \(\alpha(i, r)\). Intuitively, \(\varphi\) is \((I, \alpha)\)-typed if it honors the typing on the database and produces a typed view.

Given a database \(B=(A, R_1, \ldots, R_m)\), the view \(V=\varphi(B)\) is the database \((A, P)\), where \(P\) is the \(n\)-ary relation

\[ \{<a_1, \ldots, a_n>: B \models \varphi(a_1, \ldots, a_n)\}. \]

We use the language \(L(P)\) to talk about \(V\). When \(\varphi\) is \((I, \alpha)\)-typed, then it induces a typing \((I, \beta)\) on \(L(P)\), where \(\beta(t)\) is the type of \(x_i\), for \(1 \leq i \leq n\).

Let \(\Gamma\) be a class of databases, then \(\varphi\) defines a class of views \(\varphi(\Gamma)=\{\varphi(B): B \in \Gamma\}\). Suppose now that \(\Gamma\) is axiomatizable by \(\phi\)-dependencies (for some appropriate \(\phi\)). What can we say about \(\varphi(\Gamma)\)? The result of the previous section enables us to answer this question for an important family of view definitions. (We consider only \(\phi\)-full in this abstract.)

A view definition is conjunctive if it of the form

\[ \forall y_1 \cdots y_k (A_1 \land \cdots \land A_p) \].
where all the \( A \)'s are relational formulas. Conjunctive view definitions have been widely investigated since their introduction in [CM]. Let now \( \varphi(x_1, \ldots, x_n) \) be a view definition, and let \( (I, \alpha) \) be a typing for \( L(R_1, \ldots, R_m) \). We say that \( \varphi \) is \((I, \alpha)\)-simple if it is conjunctive, \((I, \alpha)\)-typed, and whenever \( R_i(v_1, \ldots, v_r, \ldots, v_n) \) occurs in \( \varphi \), and there is a free variable \( x_j \) whose type is \( \alpha(i, r) \), then \( \psi_r \) is \( x_j \). Several important database operations like projection and natural join can be expressed by simple conjunctive typed view definitions [ASU].

**Theorem 6.** Let \( (I, \alpha) \) be a typing for \( L(R_1, \ldots, R_m) \), let \( D \) be a set of \((I, \alpha)\)-typed full dependencies of \( L(R_1, \ldots, R_m) \), let \( \Gamma = \text{SAT}(D) \), let \( \varphi(x_1, \ldots, x_n) \) be an \((I, \alpha)\)-simple view definition, and let \( (I, \beta) \) be the typing on \( L(P) \) induced by \( \varphi \). Then \( \varphi(\Gamma) \) is axiomatisable by \((I, \beta)\)-typed full dependencies of \( L(P) \).

**Outline of Proof.** We need only to show that \( \varphi(\Gamma) \) satisfies the conditions of Theorem 4. It is easy to see that, since \( \varphi \) is conjunctive, if \( B \) is trivial then so is \( \varphi(B) \). Also, it is easy to see that \( \varphi(\Gamma) \) is domain independent. Now using the fact that \( D \) and \( \varphi \) are \((I, \alpha)\)-typed we can show that \( \varphi(\Gamma) \) is closed under \((I, \beta)\)-renamings. To see that \( \varphi(\Gamma) \) is closed under direct-products, consider the class \( \Delta \) of extended databases \( (A, R_1, \ldots, R_m, P) \), where \( (A, R_1, \ldots, R_m) \) is in \( \Gamma \) and \( P = \varphi(A, R_1, \ldots, R_m) \). Now, \( \Delta = \text{SAT}(D \cup (\varphi)) \), where \( \sigma \) is the sentence

\[
\forall y_1, \ldots, y_n (P(y_1, \ldots, y_n) \iff \varphi(y_1, \ldots, y_n)).
\]

\( \sigma \) is equivalent to the conjunction of dependencies; so by Lemma 5 we have that \( \Delta \) is closed under direct products. It follows that \( \varphi(\Gamma) \) is closed under direct products. Finally, using the facts that \( \varphi \) is \((I, \alpha)\)-simple and \( \Gamma \) is closed under \((I, \alpha)\)-renamings and subdatabases (by Lemma 4), we show that \( \varphi(\Gamma) \) is closed under subdatabases.

Since projection and natural join can be defined by a simple conjunctive typed definition, Theorem 5 (and its analog for full tgd's and egd's) implies the axiomatizability results in [Fag2] and [Hu]. Observe that the typedness condition in the theorem is required not only to ensure that \( \varphi(\Gamma) \) is closed under \((I, \alpha)\)-renamings, but also to ensure that it is closed under subdatabases.

Our second application is the axiomatizability of consistent databases. It has been argued (e.g., [U]) that unsophisticated users may not be able to deal with the database directly. Rather, these users should deal with a virtual database consisting of a single relation, called the universal relation. Thus, the semantics of the database is defined via that universal relation [Ho,GM].

Let us fix a typing \((I, \alpha)\) for \( L(R_1, \ldots, R_m) \) with \( |I| = \eta \). The universal relation \( P \) is \( \eta \)-ary, and we use the language \( L(P) \) to talk about it. We denote by \( \text{type}(t) \) the sequence \(<\alpha(i, 1), \ldots, \alpha(i, n_i)>\), and by \( \text{type}(\bar{t}) \) the sequence complementary to \( \text{type}(t) \) with respect to \(<1, \ldots, n>\). Let \( P \) be an \( \eta \)-ary relation, and let \( i_1, \ldots, i_b \), denoted \( i \), be an increasing sequence of numbers from \( \{1, \ldots, n\} \). The projection of \( P \) on \( i \), denoted \( \pi_i(P) \), is the relation

\[
\{<a_{i_1}, \ldots, a_{i_b}> \mid <a_1, \ldots, a_n> \in P\}.
\]

A database \( B = (A, R_1, \ldots, R_m) \) is consistent with respect to a set \( D \) of sentences of \( L(P) \) if there exists a database \( (A', P) \) that satisfies \( D \) such that \( \forall_i \pi_{\text{type}(i)}(P), \) for \( 1 \leq i \leq m \). Such a database is
called a weak instance of \( B \) [Ho]. Thus, we view the database as describing partial information about the universal relation, and it is consistent if this partial information can be completed. Let us denote the class of databases that are consistent with respect to \( D \) by \( CONS(D) \) (we assume that the language \( L(R_1, \ldots, R_m) \) and the typing \((I, \alpha)\) are understood from the context).

**Theorem 7.** Let \((I, \alpha)\) be a typing for \( L(R_1, \ldots, R_m) \), and let \( D \) be a set of dependencies of \( L(P) \): The class \( CONS(D) \) is axiomatizable by egd's of \( L(R_1, \ldots, R_m) \).

Observe that the dependencies in \( D \) do not have to be full or typed. In contrast, the technique in [Fag2] seems to be applicable only to full dependencies.

We now use Theorem 5 to demonstrate a negative finite axiomatizability result.

**Example.** Let the database have two binary relations \( R_1 \) and \( R_2 \). Let \((I, \alpha)\) be a typing for \( L(R_1, R_2) \), with \( I = \{1, 2, 3\}, \alpha(1, 1) = 1, \alpha(1, 2) = 2, \alpha(2, 1) = 1, \) and \( \alpha(2, 2) = 3 \). The universal relation \( P \) is hence a ternary relation, where \( R_1 = \pi_{1,2,3}(P) \) and \( R_2 = \pi_{1,3,2}(P) \). Let \( D \) consists of the following dependencies:

\[
\forall y_1 \ldots y_5(P(y_1, y_2, y_3) \land P(y_1, y_4, y_5) \rightarrow y_2 = y_4)
\]

and

\[
\forall y_1 \ldots y_5(P(y_3, y_2, y_1) \land P(y_5, y_4, y_1) \rightarrow y_2 = y_4).
\]

We claim that \( CONS(D) \) is not local, and hence it is not finitely axiomatizable by full dependencies. To see that \( CONS(D) \) is not \( n \)-local for \( n > 0 \), consider the database \( B \) with

\[
R_1 = \{<1, 1>, <n + 1, n + 1>\}, \text{ and } R_2 = \{<i, i>: 1 \leq i \leq n\} \cup \{<i + 1, i>: 1 \leq i \leq n\}
\]

We leave it to the reader to show that \( B \) is not in \( CONS(D) \), but any database of \( B \) with at most \( n \) elements is in \( CONS(D) \).

7. **Concluding Remarks**

We have given a precise characterizations of axiomatizability and finite axiomatizability by full dependencies, full tgd's and egd's. The characterizations are given in terms of closure conditions on the database classes under consideration. This closure conditions parallel known and new preservation properties of dependencies.

Unfortunately, our theory can not deal with axiomatizability by the general family of dependencies (not necessarily full), even though several preservation properties are known for these dependencies (see Section 3 and also [CLM, Fag2]). The problem is that these properties necessarily involve infinite structures. Some negative result about axiomatizability by dependencies are given in [Va].

Technion - Computer Science Department - Technical Report CS0326 - 1984
Finally, we would like to comment that the investigation of axiomatizability completely ignores the practical aspect of recognizing databases that are members of the classes under consideration. This aspect is investigated in [GV] and in [Va].
References


