Two Applications of Equational Theories to Database Theory

Stavros S. Cosmadakis
MIT
Paris C. Kanellakis[1]
MIT

Abstract

Databases and equational theorem proving are well developed and seemingly unrelated areas of Computer Science Research. We provide two natural links between these fields and demonstrate how equational theorem proving can provide useful and tools for a variety of database tasks.

Our first application is a novel way of formulating functional and inclusion dependencies (the most common database constraints) using equations. The central computational problem of dependency implication is directly reduced to equational reasoning. Mathematical techniques from universal algebra provide new proof procedures and better lower bounds for dependency implication. The use of REVE, a general purpose transformer of equations into term rewriting systems, is illustrated on nontrivial sets of functional and inclusion dependencies.

Our second application demonstrates that the uniform word problem for lattices is equivalent to implication of dependencies expressing transitive closure, together with functional dependencies. This natural generalization of functional dependencies, which is not expressible using conventional database theory formulations, has a natural inference system and an efficient decision procedure.

1. Introduction

In order to deal formally with the problems of logical database design and data processing, database theory models data as sets of tables (*relations*). These relations are required to satisfy integrity constraints (*dependencies*), which intend to capture the semantics of a particular application. Various kinds of dependencies have been proposed in the literature (see [20, 13] for reviews of the area). For example, a *functional dependency* (FD) is a formal statement of the form EMPLOYEE→SALARY, which intuitively states that every employee has a unique salary. An *inclusion dependency* (IND) is a statement of the form MANAGER⊆EMPLOYEE, which intuitively states that every manager is an employee (the more general IND MANAGER.MANAGER-SALARY⊆EMPLOYEE.EMPLOYEE-SALARY expresses also the fact that managers make the same salary as managers as they make as employees). FD's and IND's are the most common database constraints.

A most general formulation of dependencies as sentences in first order logic (namely Horn clauses) was given in [13]. To handle the central computational problem of dependency *implication* a particular proof procedure was developed, the *chase* (see [20] for its wide applicability). Proof procedures for general data dependencies also appear in [23, 2, 3].The chase was seen to be a special case of a classical theorem proving technique, namely *resolution* [2, 3].

---

[1]On leave from Brown University; supported partly by NSF grant MCS-8210830 and partly by ONR-DARPA grant N00014-83-K-0146, ARPA Order No. 4786.

Alternative methods for theorem proving have been developed in the context of *equational theories*. This is a fragment of first order logic which has attracted a lot of attention because of its wide applicability in areas such as applicative languages, interpreters, and data types. See [15] for a survey of the area.

Given the formulation of database constraints as first order sentences, one would expect database theory to have been influenced by the developments in equational theories. However, not only did this never happen, but a constant effort has been made to minimize the role of equality in data dependencies (*multivalued dependencies*, the most widely studied after FD's, do not involve equality explicitly). This is even more impressive in view of the fact that the best algorithm for *losslessness of joins*, a basic computational problem, was derived from an efficient algorithm for *congruence closure* [12]. Also the best algorithm for implication of FD's [1] can be seen directly as a special case of an algorithm of [17] for the *generator problem in finitely presented algebras*; this last observation was made recently by the authors of this paper, and seems to have escaped the notice of the database theory community.

The *implication problem* for FD's and IND's (i.e., given a set Σ of FD's and IND's and an FD or IND σ, do all databases satisfying Σ also satisfy σ) has been studied in some detail; [6, 10, 19, 7, 16, 9]. In this paper we will use the FD,IND implication problem to illustrate a surprisingly close connection between dependency inference and equational theorem proving.

We first present a transformation of FD and IND implication into equational implication (Theorem 1). This not only reveals the underlying computational structure (Corollaries 1.1, 1.2), but also leads to a proof procedure very different from the standard technique (Theorem 2). We also demonstrate how database constraints can be compiled into rewrite rules, which can then be used to make inferences (Section 3.2). The full theoretical treatment of this transformation and its use in deriving upper and lower bounds for the implication problem appears in [10].

Our second transformation (Theorem 3) reveals the *semilattice* structure of FD's. The full treatment of this transformation requires the development of special semantics for relational databases [11]. We include it in our exposition because, despite its very different nature, it also illustrates a direct connection between database theory and equational theories. In fact, completing the semilattice into a lattice naturally provides transitive closure and *partition dependencies* to our arsenal of possible database constraints. The elegant Armstrong rules for FD implication [20] can now be seen as a special case of a proof system for equational implication in lattices (Theorem 4). This theorem also provides an efficient algorithm for the uniform word problem for lattices.

In summary, the purpose of this paper is to collect from [10, 11] the basic ideas which indicate that *equational theorem proving is the proper setting for database dependency implication*. Some experimentation in this direction has also been made, using the REVE system [14, 18]; the examples in Section 3.2 clearly demonstrate how a general-purpose equational theorem prover succesfully handles cases of nontrivial FD and IND implication.

2. Definitions

2.1. Equational Theories

Let M be a set of symbols and ARITY a function from M to the nonnegative integers $\mathcal{N}$. The set of finite strings over M is M*. Partition M into two sets:

$G = \{g \in M \mid \text{ARITY}(g) = 0\}$ the *generators*,
$O = \{\theta \in M \mid \text{ARITY}(\theta) > 0\}$ the *operators*.

The set of *terms* over M, $\mathcal{T}(M)$, is the smallest subset of M* such that,

1) every g in G is a term,
2) if $\tau_1,...,\tau_m$ are terms and θ is in O with $\text{ARITY}(\theta)=m$, then $\theta\tau_1...\tau_m$ is a term.

A *subterm* of τ is a substring of τ, which is also a term. Let $V=\{x,x_1,x_2,...\}$ be a set of *variables*. Then the set of terms over operators O and generators GUV will be denoted by $\mathcal{T}^+(M)$. For terms $\tau_1,...,\tau_k$ in $\mathcal{T}^+(M)$ we can define the substitution $\varphi=\{ (x_i \leftarrow \tau_i) \mid 1 \le i \le k \}$ to be a function from $\mathcal{T}^+(M)$ to $\mathcal{T}^+(M)$. We use $\varphi(\tau)$ or $\tau[x_1/\tau_1,...,x_k/\tau_k]$ for the result of replacing all occurences of variables x_i in term τ by term τ_i $(1 \le i \le k)$, where these changes are made simultaneously.

An *equation* e is a string of the form $\tau = \tau'$ where τ,τ' are in $\mathcal{T}^+(M)$. We use the symbol E for a set of equations. We will be dealing with models for sets of equations, i.e., algebras. We consider each equation e as a sentence of first-order predicate calculus (with equality), with all variables *universally quantified.*

An *algebra* $\mathcal{A}=(A,F)$ is a pair, where A is a nonempty set and F a set of functions. Each f in F is a function from A^n to A, for some n in $\mathcal{N}$ which we call the *type(f).*

Examples: (a) A *semigroup* $(A,\{+\})$ is an algebra with one associative binary operator, i.e., for all x,y,z in A $(x+y)+z=x+(y+z)$. An example of a semigroup is the algebra of the set of functions from $\mathcal{N}$ to $\mathcal{N}$, together with the composition operation. In semigroups we use ab instead of a+b and w.l.o.g. omit parentheses.

(b) $\mathcal{A}_M$ is an algebra with $A=\mathcal{T}(M)$. For each θ in O we define a function θ in F with $type(\theta)=\text{ARITY}(\theta)$; here we use the same symbol for the syntactic object θ and its interpretation. The function θ maps terms $\tau_1,...,\tau_m$ from $\mathcal{T}(M)$ to the term $\theta\tau_1...\tau_m$, (i.e., $\theta(\tau_1,...,\tau_m)=\theta\tau_1...\tau_m$). We will refer to $\mathcal{A}_M$ as the *free algebra* on M. From this example it is clear that we can without ambiguity use both $\theta\tau_1...\tau_m$ and $\theta(\tau_1,...,\tau_m)$ to denote the same term. One can similarly define an algebra with domain $\mathcal{T}^+(M)$.

Implication: Let e be an equation and $\mathcal{A}$ an algebra. $\mathcal{A}$ satisfies e, or is a model for e, if e becomes true when its operators and nonvariable generators are interpreted as the functions of $\mathcal{A}$ and its variables take *any* values in $\mathcal{A}$'s domain. The class of all algebras which are models for a set of equations E is called a *variety* or an *equational class.* We say that E implies e ($E \models e$) if equation e is true in every model of E. An *equational theory* is a set of equalities E (of terms over $\mathcal{T}^+(M)$), closed under *implication.*

We write E⊢e, if there exists a finite proof of e starting from E and using only the following five rules:

$\tau = \tau$,

from $\tau_1 = \tau_2$ *deduce* $\tau_2 = \tau_1$,

from $\tau_1 = \tau_2$ *and* $\tau_2 = \tau_3$ *deduce* $\tau_1 = \tau_3$,

from $\tau_i = \tau_i'$ $(1 \le i \le m)$ *deduce* $\theta\tau_1...\tau_m = \theta\tau_1'...\tau_m'$ (ARITY$(\theta)=m$),

from $\tau_1 = \tau_2$ *deduce* $\varphi(\tau_1) = \varphi(\tau_2)$ (φ is any substitution).

Proposition 1: [4] $E \models \tau = \tau'$ iff $E \vdash \tau = \tau'$. ∎

A binary relation $\approx$ on $\mathfrak{T}(M)$ or $\mathfrak{T}^+(M)$ is a *congruence* provided that,

1) $\approx$ is an equivalence relation,
2) if ARITY$(\theta)=m$ and $\tau_i \approx \tau_i'$ $(1 \le i \le m)$ then $\theta\tau_1...\tau_m \approx \theta\tau_1'...\tau_m'$.

Let $\approx$ be a congruence on $\mathfrak{T}(M)$. Congruence guarantees that the operations in O are well-defined on $\approx$-equivalence (or congruence) classes. Thus we can form a *quotient* algebra $\mathfrak{T}(M)/\approx$ with domain $\{[\tau] \mid \tau$ in $\mathfrak{T}(M)$, $[\tau]$ is the $\approx$-congruence class of $\tau\}$ and with functions corresponding to O's operators.

Let Γ be a set of equations over terms in $\mathfrak{T}(M)$ (i.e., containing no variables). Consider the equational theory consisting of all $\tau = \tau'$ such that $\Gamma \models \tau = \tau'$. By Proposition 1 this theory induces a congruence $=_\Gamma$ on $\mathfrak{T}(M)$, where $\tau =_\Gamma \tau'$ iff $\Gamma \models \tau = \tau'$. From the remark above we see that this congruence naturally defines an algebra $\mathfrak{T}(M)/=_\Gamma$. If Γ is a finite set $\mathfrak{T}(M)/=_\Gamma$ is known as a *finitely presented algebra* [17].

These are the only definitions needed to make our exposition self-contained. For an extended survey of the area, definitions of term rewriting systems and more general definitions, see [15].

2.2. Relational Database Theory

Let $\mathfrak{U}$ be a finite set of *attributes* and $\mathfrak{D}$ a countably infinite set of *values*, such that $\mathfrak{U} \cap \mathfrak{D} = \emptyset$. A *relation scheme* is an object R[U], where R is the *name* of the relation scheme and $U \subseteq \mathfrak{U}$. A *tuple* t over U is a function from U to $\mathfrak{D}$. Let A_i be an attribute in U and a_i a value, where $1 \le i \le |U|$; *if* $t[A_i] = a_i$ *then* we represent tuple t over U as $a_1a_2...a_{|U|}$. We represent the restriction of tuple t on attributes $A_1...A_n$ of U as $t[A_1...A_n]$. A *relation* r over U (named R) is a (possibly infinite) nonempty set of tuples over U. A *database scheme* D is a finite set of relation schemes $\{R_1[U_1],....R_q[U_q]\}$ and a *database* $d = \{r_1,...,r_q\}$ associates each relation scheme $R_i[U_i]$ in d with a relation r_i over U_i. A database is finite if all of its relations are finite. A database can be visualized as a set of tables, one for each relation, whose headers are the relation schemes (each column headed by an attribute), and whose rows are the tuples.

The logical constraints, which determine the set of legal databases, are called *database dependencies*. We will be examining two very common types of dependencies. These are sentences over relation names and attributes, which are either satisfied or falsified by relations.

FD $R: A_1...A_n \rightarrow A$ is a *functional dependency*.

Relation r (named R) satisfies this FD if, for tuples t_1, t_2 in r, $t_1[A_1...A_n] = t_2[A_1...A_n]$ implies $t_1[A] = t_2[A]$. If $n=1$ we call the dependency a *unary functional dependency* (uFD).

It is quite common in the database literature to use the notation $R:A_1...A_n \rightarrow AB$ for $R:A_1...A_n \rightarrow A$ and $R:A_1...A_n \rightarrow B$ (i.e., for more than one FD with the same left-hand side).

IND $S:C_1...C_m \subseteq R:B_1...B_m$ is an *inclusion dependency.*
Relations s,r (named S,R respectively) satisfy this IND if, for each tuple t in s, there is a tuple t_1 in r with $t_1[B_i]=t[C_i]$ for $1 \leq i \leq m$. If $m=1$ we call the dependency a *unary inclusion dependency* (uIND).

Equality of two columns headed by attributes A, B in a relation named R can be expressed as a special case of IND's: use $R:AB \subseteq R:AA$. These dependencies are particularly illustrative of our analysis; we will use $A \equiv B$ to denote them.

Implication: We say that the set of dependencies Σ implies dependency σ ($\Sigma \models \sigma$) if, whenever a database d over scheme D satisfies Σ, it also satisfies σ. If we restrict ourselves to finite databases we have $\Sigma \models_{fin} \sigma$. Clearly if $\Sigma \models \sigma$ (*implication*) then $\Sigma \models_{fin} \sigma$ (*finite implication*), but the converse is not always true. Deciding implication of dependencies is a central problem in database theory. Since dependencies are sentences in first-order predicate calculus with equality, we have *proof procedures* for the implication problem (we denote proofs as $\Sigma \vdash \sigma$). A proof procedure is *sound* if when $\Sigma \vdash \sigma$ then $\Sigma \models \sigma$; and *complete* if it is sound and when $\Sigma \models \sigma$ then $\Sigma \vdash \sigma$ (similarly for finite implication). The standard complete proof procedure for database dependencies is the *chase.* We now present the chase for FD's and IND's.

Chase: Given a set of dependencies Σ over scheme D and a dependency σ, construct a set of tables T with D's relation schemes as headers. These tables are originally empty and will be filled with symbols from the countably infinite set $\mathfrak{D}$. Whenever we insert a new row of symbols from $\mathfrak{D}$ in a table of T and we do not specify some of the entries of this row, then we assume that distinct symbols from $\mathfrak{D}$, which have not yet appeared elsewhere in T, are used to fill these entries. We use t_i^r for the ith row of table R and $t_i^r[X]$ for this row's entries in the columns of attributes X.
The *initial configuration* of T depends on σ as follows:
(i) If $\sigma = R:A_1...A_n \rightarrow A$, insert rows t_1^r, t_2^r with the only restriction that $t_1^r[A_i]=t_2^r[A_i]$, where $1 \leq i \leq n$.
(ii) If $\sigma = S:C_1...C_m \subseteq R:B_1...B_m$, insert t_1^s.
Every dependency in Σ produces a *rule.* If f is an FD in Σ the corresponding FD-rule is:
<Consider T a database over symbols in $\mathfrak{D}$. If T does not satisfy f, because two symbols x and y are different then replace y by x in T>.
If $i = S:X \subseteq R:Y$ is an IND in Σ the corresponding IND-rule is:
<Consider T a database over symbols in $\mathfrak{D}$. If T does not satisfy i, because some $t^s[X]$ does not appear in the table R as some $t^r[Y]$, then insert t^r in R with $t^r[Y]=t^s[X]$.>

We will say that $\Sigma \vdash_{chase} \sigma$, if there is a finite sequence of applications of the FD-rules and IND-rules produced by Σ that transforms T's initial configuration to a final configuration satisfying:
(i) If $\sigma = R:A_1...A_n \rightarrow A$, then $t_1^r[A]=t_2^r[A]$
(ii) If $\sigma = S:C_1...C_m \subseteq R:B_1...B_m$, then $t_i^s[C_i]=t_j^r[B_i]$, where $1 \leq i \leq m$, for some j.

Proposition 2: $\Sigma \vdash_{chase} \sigma$ iff $\Sigma \models \sigma$. ∎

3. Functional and Inclusion Dependencies as Equations

3.1. Basic Transformation

Let Σ be a set of FD's and IND's over a database scheme D and σ an FD or IND. We will transform Σ into two sets of equations E_Σ and $\mathcal{E}_\Sigma$ such that the following holds: $\Sigma \models \sigma$ iff $E_\Sigma \models E_\tau$ iff $\mathcal{E}_\Sigma \models \mathcal{E}_\tau$, for sets of equations $E_\tau, \mathcal{E}_\tau$ whose form depends on Σ and σ. We assume that D only contains one relation scheme; this simplifies notation, and there is no loss of generality.

Transformation: From the dependencies in Σ construct the following sets of symbols,

$M_f = \{f_k \mid$ for each FD with an n attribute left-hand side include one operator f_k of ARITY n$\}$,
$M_i = \{i_k \mid$ for each IND include one operator i_k of ARITY 1$\}$,
$M_a = \{a_k \mid$ for each attribute A_k include one operator a_k of ARITY 1$\}$,
$M_\alpha = \{\alpha_k \mid$ for each attribute A_k include one generator $\alpha_k\}$.

Now let $M = M_f \cup M_i \cup M_a \cup M_\alpha$ and $V = \{x, x_1, x_2, \ldots\}$ be a set of variables. $\mathcal{T}^+(M_f)$ ($\mathcal{T}^+(M_i)$) are the sets of terms constructed using operators in M_f (M_i) and generators in V.

The set E_Σ consists of the following equations

1) one equation for each $\sigma_k = A_1 \ldots A_n \rightarrow A$: $f_k a_1 x \ldots a_n x = ax$,

2) m equations for each $\sigma_k = B_1 \ldots B_m \subseteq A_1 \ldots A_m$: $a_1 i_k x = b_1 x$ and ... and $a_m i_k x = b_m x$.

The set $\mathcal{E}_\Sigma$ consists of the following equations:

3) one equation for each $\sigma_k = A_1 \ldots A_n \rightarrow A$: $f_k \alpha_1 \ldots \alpha_n = \alpha$,

4) m equations for each $\sigma_k = B_1 \ldots B_m \subseteq A_1 \ldots A_m$: $i_k \alpha_1 = \beta_1$ and ... and $i_k \alpha_m = \beta_m$,

5) for each pair of symbols f_p in M_f and i_q in M_i the equation $f_p i_q x_1 \ldots i_q x_n = i_q f_p x_1 \ldots x_n$ (ARITY$(f_p) = n$).

The transformation is illustrated in Figure 1. Note that in $\mathcal{E}_\Sigma$ only equations 5) contain variables. Equations 5) are *commutativity* conditions between f and i operators. The transformation can also be thought of as the Skolemization of the definitions from Section 2.2. We now present Theorem 1, which is central to our approach. A slightly more general version of the Theorem is presented in [10].

Theorem 1: In each of the following three cases, (i),(ii),(iii) are equivalent.

$\equiv$ Case:

i) $\Sigma \models A \equiv B$

ii) $E_\Sigma \models ax = bx$

iii) $\mathcal{E}_\Sigma \models \alpha = \beta$.

FD Case:

i) $\Sigma \models A_1 \ldots A_n \rightarrow A$

ii) $E_\Sigma \models \tau[x_1/a_1 x, \ldots, x_n/a_n x] = ax$, for some τ in $\mathcal{T}^+(M_f)$

iii) $\mathcal{E}_\Sigma \models \tau[x_1/\alpha_1, \ldots, x_n/\alpha_n] = \alpha$, for some τ in $\mathcal{T}^+(M_f)$.

IND Case:

i) $\Sigma \models B_1...B_m \subseteq A_1...A_m$

ii) $E_\Sigma \models a_1\tau = b_1 x$ and ... and $a_m\tau = b_m x$, for some τ in $\mathcal{T}^+(M_i)$

iii) $\mathcal{E}_\Sigma \models \tau[x/\alpha_1] = \beta_1$ and ... and $\tau[x/\alpha_m] = \beta_m$, for some τ in $\mathcal{T}^+(M_i)$.

Proof Sketch: We use E_τ ($\mathcal{E}_\tau$) to denote the set of equations corresponding to term τ in (ii),(iii).

(ii)⇒(i) Suppose $E_\Sigma \models E_\tau$, and let relation r satisfy Σ; we will show that r satisfies σ. Relation r is, by definition, nonempty and its entries can be w.l.o.g. positive integers. Number its tuples 1,2,... etc., (it could contain a countably infinite number of tuples). Define $A(.): \mathcal{N} \to \mathcal{N}$, such that, if x is the number of a tuple in r, then A(x) is the entry in tuple x at attribute A, else A(x) is 0 ($\mathcal{N}$ are the nonnegative integers). If f is the FD $C_1...C_k \to C$ in Σ define $F(...): \mathcal{N}^k \to \mathcal{N}$, such that, if x is the number of a tuple in r, then $F(C_1(x),...,C_k(x)) = C(x)$, else F is 0. This is a well defined function since r satisfies f. If i is the IND $D_1...D_k \subseteq C_1...C_k$ in Σ define $I(.): \mathcal{N} \to \mathcal{N}$, such that, if x is the number of a tuple in r and x′ is the number of the first tuple in r where $t_x[D_1...D_k] = t_{x'}[C_1...C_k]$, then $I(x) = x'$, else I(x) is 0. This is also a well defined function since r satisfies i. We have constructed an algebra with domain $\mathcal{N}$ and functions A(.),...,F(...),...,I(.),..., which, as is easy to verify, is a model for E_Σ. Let σ be an IND. By interpreting each symbol in τ as an I(.), we see that when x is a tuple number $\tau(x)$ is another tuple number. Since $E_\Sigma \models E_\tau$, we must have $A_i(\tau) = B_i(x)$ $1 \le i \le m$, which means that r satisfies σ. The case of an FD is similar.

(iii)⇒(ii) Suppose $\mathcal{E}_\Sigma \models \mathcal{E}_\tau$, and let $\mathcal{M}$ be a model of E_Σ; we will show that $\mathcal{M}$ satisfies E_τ. From $\mathcal{M}$ we will construct a model $\mathcal{A}(\mathcal{M})$ for $\mathcal{E}_\Sigma$. The algebra $\mathcal{A}(\mathcal{M})$ will have domain all functions from $\mathcal{M}$ to $\mathcal{M}$, i.e., $\mathcal{M} \to \mathcal{M}$. In $\mathcal{A}(\mathcal{M})$ the interpretation of α will be the function $\alpha(x)$, which is the interpretation of a(.) in $\mathcal{M}$. The interpretation of i(.) will be the function $\lambda h.h(i(x))$, where $i(x)$ is the interpretation of i(.) in $\mathcal{M}$ (this is a function from $\mathcal{M} \to \mathcal{M}$ to $\mathcal{M} \to \mathcal{M}$). The interpretation of f(...) will be the function $\lambda h_1...h_n.f(h_1(x),...,h_n(x))$, where $f(x_1,...,x_n)$ is the interpretation of f(...) in $\mathcal{M}$ (this is a function from $(\mathcal{M} \to \mathcal{M})^n$ to $\mathcal{M} \to \mathcal{M}$). It is straightforward to check that equations 3),4) hold in $\mathcal{A}(\mathcal{M})$, because $\mathcal{M}$ is a model for E_Σ. Also equations 5) hold in $\mathcal{A}(\mathcal{M})$: For example, if n=1 the interpretation of f(i(h)) in $\mathcal{A}(\mathcal{M})$ is $f(h(i(x))$, which is also the interpretation of i(f(h)) (h is any element of $\mathcal{M} \to \mathcal{M}$). Thus $\mathcal{A}(\mathcal{M})$ is a model for $\mathcal{E}_\Sigma$. Since $\mathcal{E}_\Sigma \models \mathcal{E}_\tau$, $\mathcal{A}(\mathcal{M})$ satisfies $\mathcal{E}_\tau$, and it easily follows that $\mathcal{M}$ satisfies E_τ.

(i)⇒(iii) By induction on the number of steps of a chase proof of σ from Σ (see [8]). ∎

A proof procedure for IND's and FD's which is different from the chase is given in [19]. We can show that each of the rules in [19] can be simulated using the equational reasoning of Proposition 1, and thus give an alternate proof of the (i)⇒(iii) step. Let us illustrate this with an example: From A→B and CD⊆AB the *pullback rule* of [19] derives C→D. In equational language $f\alpha = \beta$, $i\alpha = \gamma$, $i\beta = \delta$ and $fix = ifx$ imply $f\gamma = fi\alpha = if\alpha = i\beta = \delta$.

Corollary 1.1: Let Σ be a set of FD's and σ an FD. The implication problem $\Sigma \models \sigma$ is equivalent to a *generator problem for a finitely presented algebra* [17].

Proof: $\mathcal{E}_\Sigma$ is now a finite set of equations with no variables. If $\approx$ is the congruence induced by $\mathcal{E}_\Sigma$ on $\mathcal{T}(M)$ then $\mathcal{T}(M)/\approx$ is a finitely presented algebra. The equational implication in Theorem 1 is known, in this case, as a generator problem for the finitely presented algebra $\mathcal{T}(M)/\approx$. ∎

Using Corollary 1.1, one can observe that the linear time algorithm of [1] for FD inference can be derived in a straightforward way from the general algorithm of [17] for the generator problem.

For the special case of uFD's and IND's, the equations resulting from our transformation can be viewed as *string equations*:

Semigroup Transformation: Let Σ be a set of IND's and uFD's. Produce the set of symbols M_s from M as follows: for each $f_k(.)$ in M_f add one generator f_k in M_s; for each $i_k(.)$ in M_i add one generator i_k in M_s; for each $a_k(.)$ in M_a add one generator a_k in M_s; add one binary operator + in M_s.
E_S consists of the associative axiom for + and the following word (string) equations (we omit + and parentheses):
1) one equation for each uFD $\sigma_k = A_1 \rightarrow A$: $f_k a_1 = a$
2) m equations for each IND $\sigma_k = B_1...B_m \subseteq A_1...A_m$: $a_1 i_k = b_1$ and ... and $a_m i_k = b_m$.

Corollary 1.2: Let Σ be a set of uFD's and IND's
$\Sigma \models A \equiv B$ iff $E_S \models a = b$
$\Sigma \models A_1 \rightarrow A$ iff $E_S \models wa_1 = a$, for some string w in M_s^*
$\Sigma \models B_1...B_m \subseteq A_1...A_m$ iff $E_S \models a_1 w = b_1$ and ... and $a_m w = b_m$, for some string w in M_s^*. ∎

Note that the first case is an instance of the *uniform word problem for semigroups*. The other two cases are known as E_S-*unification* problems [15]. Corollary 1.2 is used in [10] to study the computational complexity of the inference problem for FD's and IND's.

The approach of Theorem 1 can in fact be extended to more general database dependency statements [10, 8]. What we present here is the application of this transformation to the most practical database constraints, in order to illustrate the basic ideas involved.

3.2. Dependency Inference

In this section we will be dealing with the inference problem for IND's and uFD's. As demonstrated in [19, 7, 10] this is an undecidable problem, with a number of interesting decidable subcases [10]. We will first present a proof procedure for IND's and uFD's, which differs from the chase because it is based on equational reasoning. This procedure also differs from that of [19], because of the simplicity and symmetry of its rules. We use the graph notation developed in [9, 10].

Graph Notation: Let Σ be a set of IND's and uFD's over relation scheme R (it is very simple to generalize both the notation and the proof procedure to handle several relation schemes). We construct a labeled directed graph G_Σ, which has exactly one node a_i for each attribute A_i in R. Let $i = B_1...B_m \subseteq A_1...A_m$ be an IND in Σ. Then G_Σ contains m *black* arcs $(a_1,b_1),...,(a_m,b_m)$, each arc labeled by the name i of the IND. Let $f = A \rightarrow B$ be a uFD in Σ. Then G_Σ contains one *red* arc (a,b) labeled by the name f of the uFD.

The graph notation is illustrated in the two examples of Figure 2.

The Proof Procedure G: Given a set Σ of uFD's and IND's construct their graphical representation G_Σ. Each attribute in Σ is associated with one of the nodes of G_Σ.

Rules: Apply some finite sequence of the graph manipulation rules 1,2,3 and 4 of Figure 3 on G_Σ. Rules 1 and 2 introduce new unnamed nodes. Rules 3 and 4 identify two existing nodes; the node resulting from this identification is associated with the union of the two sets of attribute names, that were associated with each of the identified nodes. Note that rules 1,2 w.l.o.g. need be applied at most once to every left-hand side configuration.

Let G be the resulting graph.

We say that $\Sigma \vdash_G \sigma$ when:

σ is $A \equiv B$ and A,B are associated with the same node;

σ is a uFD $A \rightarrow B$ and there is a path of red arcs in G starting at A and ending at B;

σ is an IND $B_1 \ldots B_m \subseteq A_1 \ldots A_m$ and there are m black directed paths in G, all with the same sequence of labels, path i starting at A_i and ending at B_i.

Theorem 2: $\Sigma \models \sigma$ iff $\Sigma \vdash_G \sigma$.

Proof Sketch: We outline the proof for σ being $A \equiv B$.

($\Leftarrow$): Rules 3,4 are obviously sound. Rules 1 and 2 are sound in the sense of the *attribute introduction rule* of [19], which we illustrate as rule 5 of Figure 3.

($\Rightarrow$): We assume that we cannot prove σ, and construct a model for $\mathcal{E}_\Sigma$ in which $\alpha \neq \beta$; then by Theorem 1 Σ does not imply σ. If σ is not provable, then there is a (possibly infinite) graph G which represents Σ, is closed under the rules, and in which the names A and B correspond to different nodes. We add one special node $\perp$ to G. The labels of G are symbols corresponding to INDs (i symbols) or uFDs (f symbols) of Σ. If a node in $G \cup \{\perp\}$ has no outgoing arc labeled with some i, add one going to $\perp$. Repeat for the f symbols. The resulting graph represents functions interpreting the operators and generators in $\mathcal{E}_\Sigma$: This is because closure with respect to rules 3 and 4 and the padding of G we performed, guarantees functionality. The node A (B) is the interpretation of α (β). Now closure with respect to rules 1 and 2 guarantees the commutativity conditions of $\mathcal{E}_\Sigma$, and the fact that G represents Σ guarantees equations 3),4) of $\mathcal{E}_\Sigma$. Thus there is a model of $\mathcal{E}_\Sigma$ in which $\alpha \neq \beta$. ∎

A different (*heuristic*) approach to dependency inference could proceed as follows: We first translate the dependencies into equations using the Semigroup Transformation of Section 3.1. We then try to compile them into a (confluent and noetherian) rewrite rule system using some generalized Knuth-Bendix type procedure. If we succeed, the resulting rewrite rule system would be very useful in making many of the desired inferences.

We demonstrate this general approach on three examples. Each example illustrates some of the inherent difficulties of the FD and IND implication problem. The REVE system [14, 18] has been used in all cases. See [15] for rewrite rule notations and definitions.

Example 1: Consider the uFD's and uIND's in Figure 2a. As shown in [16], uFD and uIND statements can be handled independently (i.e., they do not interact). However, implication is still non-trivial because it differs from *finite* implication (which, nonetheless, can also be decided in polynomial time [16]). The instances which differentiate between finite and unrestricted implication are exactly those containing mixed cycles of uIND's and uFD's, such as the one in Figure 2a. We translated these statements into equations and compiled them into the following rewrite rule system, using REVE:

Rules:
(fa) → b ((xf)a) → (xb)
(bi) → c ((xb)i) → (xc)
(gc) → d ((xg)c) → (xd)
(bj) → a ((xb)j) → (xa)
(x(yz)) → ((xy)z)

As expected, the rules clearly show that the uIND and FD statements decouple.

Example 2: Consider the dependencies f=C→D, i=AB⊆CD, j=BA⊆CD, k=B⊆A. This is the example used in [19] to illustrate the necessity of the *attribute introduction* rule. REVE succeeded in compiling the dependencies into the following rewrite rule system:

Rules:
(fc) → d ((xf)c) → (xd)
(ci) → a ((xc)i) → (xa)
(di) → b ((xd)i) → (xb)
(dj) → a ((xd)j) → (xa)
(cj) → b ((xc)j) → (xb)
(ak) → b ((xa)k) → (xb)
(x(yz)) → ((xy)z)

(fa) → b ((xf)a) → (xb)
(fb) → a ((xf)b) → (xa)
(bk) → a ((xb)k) → (xa)

The non-trivial inference made in this case is that AB⊆BA follows from the dependencies in Figure 2b: observe the "inferred" rule (bk) → a.

Example 3: The dependencies in Figure 2b correspond to the base case of the proof in [9] that certain restricted proof procedures cannot exist for FD's and IND's (the argument uses the chase procedure in a structured fashion). REVE compiled these dependencies into the following rewrite rule system:

Rules:

$(fa_1) \rightarrow c_1$ $((xf)a_1) \rightarrow (xc_1)$
$(gc_2) \rightarrow b_2$ $((xg)c_2) \rightarrow (xb_2)$
$(hd_2) \rightarrow b_2$ $((xh)d_2) \rightarrow (xb_2)$
$(c_2i) \rightarrow c_1$ $((xc_2)i) \rightarrow (xc_1)$
$(d_2i) \rightarrow d_1$ $((xd_2)i) \rightarrow (xd_1)$
$(a_1j) \rightarrow a_3$ $((xa_1)j) \rightarrow (xa_3)$
$(d_1j) \rightarrow d_3$ $((xd_1)j) \rightarrow (xd_3)$
$(d_2k) \rightarrow d_3$ $((xd_2)k) \rightarrow (xd_3)$
$(b_2k) \rightarrow b_3$ $((xb_2)k) \rightarrow (xb_3)$
$(x(yz)) \rightarrow ((xy)z)$

$(hd_3) \rightarrow b_3$ $((xh)d_3) \rightarrow (xb_3)$
$(b_2i) \rightarrow (hd_1)$ $((xb_2)i) \rightarrow ((xh)d_1)$
$(c_1j) \rightarrow (fa_3)$ $((xc_1)j) \rightarrow ((xf)a_3)$
$(gc_1) \rightarrow (hd_1)$ $((xg)c_1) \rightarrow ((xh)d_1)$
$((gf)a_3) \rightarrow b_3$ $(((xg)f)a_3) \rightarrow (xb_3)$

The non-trivial inference here is that $A_3 \rightarrow B_3$ follows from the dependenies of Figure 2c: observe the "inferred" rule $((gf)a_3) \rightarrow b_3$.

4. Functional Dependencies, Transitive Closure and Lattices

4.1. Functional Dependency Inference Revisited

The implication problem for functional dependencies was one of the first computational problems identified as central to database theory [1]. As we saw in Corollary 1.1, it is directly related to the generator problem for finitely presented algebras. In this section we will reformulate it as a word problem in a lattice, which will reveal much more of its algebraic structure.

Let $+, \cdot$ be two operators satisfying the *lattice axioms* (LA):

1. $x+x=x$, $x \cdot x=x$ (idempotency)
2. $x+y=y+x$, $x \cdot y=y \cdot x$ (commutativity)
3. $x+(y+z)=(x+y)+z$, $x \cdot (y \cdot z)=(x \cdot y) \cdot z$ (associativity)
4. $x+(x \cdot y)=x$, $x \cdot (x+y)=x$ (absorption)

Let us assume, again for simplicity, that we have one relation scheme R[U] with attributes U. For every attribute A in U, introduce a constant symbol A. Using these constants and the operators $+,\bullet$ we can form expressions, which we call *partition expressions.* An equation $p=q$ involving two partition expressions is a *partition dependency* (PD). For example, $A=A\bullet B$, $A+B=C$ are PD's.

In [11] we develop the semantics of PD's and show that they are a proper generalization of FD's. Let E be a set of PD's and e a PD: $E\models_{lat} e$ iff e is implied from EULA.

ACI-Transformation: Given a set of FD's F we transform them into a set of equations E_F as follows:
$f=A_1A_2...A_n\rightarrow A$ is transformed into e_f: $A_1\bullet A_2\bullet...\bullet A_n=A\bullet A_1\bullet A_2\bullet...\bullet A_n$, where $\bullet$ is associative, commutative and idempotent.

Theorem 3 [11]: $F\models f$ iff $E_F\models_{lat} e_f$ ∎

In fact, this is a more useful approach than the generator formulation of Corollary 1.1. The implication problem for FD's can thus be reduced, in this straightforward way, to the (uniform) *word problem for semilattices* (structures with a single associative, commutative and idempotent operator). On the other hand, since $X=Y$ is equivalent to $X=X\bullet Y$ *and* $Y=Y\bullet X$, we can also reduce the above word problem to the implication problem for FD's.

In order to complete our exposition we would like to comment on the semantics of a PD which contains the + operator (for FD's we have been using $\bullet$ only).
From [11], it follows that a relation r satisfies the PD $C=A+B$ when, for any tuples $t,s\in r$,
$t[C]=s[C]$ iff there are tuples $s_0,...,s_n$ of r with $t=s_0$, $s_n=s$, and for $i=0,...,n-1$ $s_i[A]=s_{i+1}[A]$ *or* $s_i[B]=s_{i+1}[B]$.

Example: Consider a database d with only one relation r representing an undirected graph. This relation has three attributes: HEAD, TAIL and COMPONENT. For every edge {a,b} in the graph we have in the relation tuples abc, bac, aac, bbc, where c is a number which could vary with a and b. These are the only tuples in r. We would like to express that: *component is the connected component in which the arc (head, tail) belongs.* We can do this by enforcing the PD COMPONENT = HEAD + TAIL.

4.2. On the Uniform Word Problem for Lattices

We have seen how FD implication is equivalent to the uniform word problem for semilattices (idempotent commutative semigroups). We have also motivated a larger class of dependencies, partition dependencies, which can naturally express transitive closure. The problem of partition dependency implication is equivalent to the *uniform word problem for lattices.* This problem is studied in [11]. From the analysis in [11] it follows that there is a polynomial-time algorithm for this word problem, and therefore for PD implication. Specifically, one has the following proof procedure for implication:

Theorem 4 [11]: $E \models_{lat} p=q$ iff $p \leq_E q$ and $q \leq_E p$ can be proved using the following rules:

1. $A \leq_E A$, A in $\mathcal{U}$.
2. $z \leq_E w$, $w \leq_E z$ for $z=w$ in E.
3. *from* $p \leq_E q$, $q \leq_E r$ *derive* $p \leq_E r$.
4. *from* $p \leq_E r$, $q \leq_E r$ *derive* $p+q \leq_E r$.
5. *from* $p \leq_E r$ *derive* $p \cdot q \leq_E r$.
6. *from* $r \leq_E p$, $r \leq_E q$ *derive* $r \leq_E p \cdot q$.
7. *from* $r \leq_E p$ *derive* $r \leq_E p+q$. ∎

The above proof procedure directly leads to a polynomial-time algorithm for PD implication: Observe that, if there is a proof that $p \leq_E q$, then this proof need only mention subexpressions of p, q, and of the expressions appearing in E. Thus, we can just write down these expressions (say, as in [17]) and repeatedly apply the rules, until no new inference can be made.

If E is empty we obtain a special case of the uniform word problem, namely that of recognizing *identities*. It had been shown [22] that rules 1, 4-7 above form a complete inference system for identities. We also remark that the proof procedure of Theorem 4 is a generalization of the Armstrong rules for FD's [20].

Since inference of FD's can be seen as a special case of inference of PD's, the problem is actually *logspace complete* for PTIME [21]. However, in the special case where E is empty (i.e. the identities) it can be solved in *logarithmic space* as follows: we first rewrite $p=q$ as a Boolean tree with leaves of the form $A \leq B$, A,B in $\mathcal{U}$. We then replace $A \leq A$ by *true* and $A \leq B$ by *false* if $A \neq B$, and evaluate the resulting tree.

Example: $A+B=C \cdot D$ is (recursively) rewritten as

$$A+B \leq C \cdot D \wedge C \cdot D \leq A+B$$

$$(A+B \leq C \wedge A+B \leq D) \wedge (C \cdot D \leq A \vee C \cdot D \leq B)$$

$$((A \leq C \wedge B \leq C) \wedge (A \leq D \wedge B \leq D)) \wedge ((C \leq A \vee D \leq A) \vee (C \leq B \vee D \leq B))$$

5. Conclusions

We have shown how the problem of FD,IND implication can be transformed to an equational implication problem (Theorem 1). Our approach can in fact be generalized to handle the implication problem for the most general dependencies of [13] (see [8]). We have solved the uniform word problem for lattices in polynomial time; for distributive lattices, this problem becomes NP-hard [5].

We have experimented with REVE on FD and IND implication. We believe that an equational theorem prover based on ACI-unification could be useful for the problems outlined in Section 4.

A shortcoming of our approach is that it cannot directly handle *finite* implication. A number of positive results in that area are contained in [10, 16].

Acknowledgement

We would like to thank the REVE group at M.I.T., and in particular John Guttag, Kathy Yelick and Dave Detlefs.

References

1. Beeri, C. and Bernstein, P.A. "Computational Problems Related to the Design of Normal Form Relational Schemas". *ACM Transactions on Database Systems 4*, 1 (March 1979), 30-59..

2. Beeri, C. and Vardi, M.Y. "Formal Systems for Tuple and Equality Generating Dependencies". *SIAM Journal of Computing 13*, 1 (February 1984), 76-98..

3. Beeri, C. and Vardi, M.Y. "A Proof Procedure for Data Dependencies". *Journal of the Association for Computing Machinery 31*, 4 (October 1984), 718-741..

4. Birkhoff, G. "On the Structure of Abstract Algebras". *Proceedings of the Cambridge Philosophical Society 31*, (1935).

5. Bloniarz, P.A., Hunt, H.B. III and Rosenkrantz, D.J. "Algebraic Structures with Hard Equivalence and Minimization Problems". *Journal Of The ACM 31*, 4 (October 1984), 879-904..

6. Casanova, M.A., Fagin, R. and Papadimitriou, C.H. "Inclusion Dependencies and Their Interaction with Functional Dependencies". *Journal of Computer and System Sciences 28*, 1 (February 1984), 29-59..

7. Chandra, A.K. and Vardi, M.Y. The Implication Problem for Functional and Inclusion Dependencies is Undecidable. IBM Tech. Rep. RC 9980, , , 1983.

8. Cosmadakis, S.S. *Equational Theories and Database Constraints.* Ph.D. Th., Massachusetts Institute of Technology, 1985.

9. Cosmadakis, S.S. and Kanellakis, P.C. "Functional and Inclusion Dependencies: A Graph Theoretic Approach". *Proceedings of the 3rd ACM Symposium on Principles of Database Systems* (April 1984), 24-37.

10. Cosmadakis, S.S. and Kanellakis, P.C. "Equational Theories and Database Constraints". *Proceedings of the 17th Annual ACM Symposium on Theory of Computing* (May 1984),.

11. Cosmadakis, S.S., Kanellakis, P.C. and Spyratos, N. "Partition Semantics for Relations". *Proceedings of the 4th ACM Symposium on Principles of Database Systems* (March 1985),.

12. Downey, P.J., Sethi, R. and Tarjan, R.E. "Variations on the Common Subexpression Problem". *Journal of the Association for Computing Machinery 27*, 4 (October 1980), 758-771..

13. Fagin, R. "Horn Clauses and Database Dependencies". *Journal of the ACM 29*, 4 (October 1982), 952-985..

14. Forgaard, R. and Guttag, J.V. "REVE: A Term Rewriting System Generator with Failure Resistant Knuth-Bendix". *Proceedings of an NSF Workshop on the Rewrite Rule Laboratory* (April 1984), 5-31.

15. Huet, G. and Oppen, D. Equations and Rewrite Rules: a Survey. In *Formal Languages: Perspectives and Open Problems*, Eds., Academic Press, , 1980.

16. Kanellakis, P.C., Cosmadakis, S.S. and Vardi, M.Y. "Unary Inclusion Dependencies Have Polynomial Time Inference Problems". *Proceedings of the 15th Annual ACM Symposium on Theory of Computing* (1983).

17. Kozen, D. "Complexity of Finitely Presented Algebras". *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, ACM SIGACT* (May 1977), .

18. Lescanne, P. "Computer Experiments with the REVE Term Rewriting System Generator". *Proceedings of the 10th ACM Symposium on Principles of Programming Languages* (January 1983), 99-108.

19. Mitchell, J.C. "The Implication Problem for Functional and Inclusion Dependencies". *Information and Control 56*, 3 (March 1983), 154-173. .

20. Ullman, J.D.. *Principles of Database Systems.* Computer Science Press, Inc., , 1983.

21. Vardi, M.Y. "Personal Communication". ( ).

22. Whitman, P.M. "Free Lattices". *Annals of Mathematics 42*, (1941).

23. Yannakakis, M. and Papadimitriou C.H. "Algebraic Dependencies". *J. Comput. Systems Sci. 21*, 1 (August 1982), 2-41. .

Σ : $A \to B$, $C \to D$
$CD \leq AB$
$A'B' \to C'$

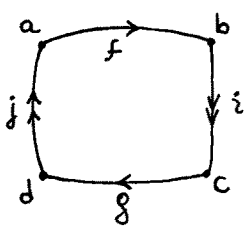
E_Σ :
$f\alpha x = bx$
$g c x = d x$
$a i x = c x$
$b i x = d x$
$f' \alpha' x b' x = C' x$

E_Σ :
$f\alpha = \beta$
$g\gamma = \delta$
$i\alpha = \gamma$
$i\beta = \delta$
$f'\alpha'\beta' = \gamma'$

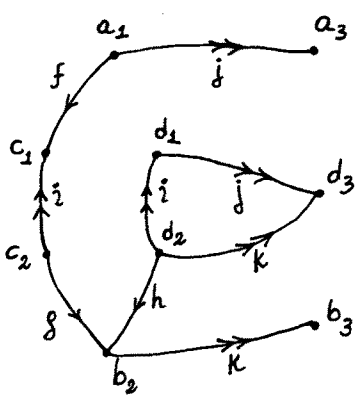
$fix = ifx \quad gix = igx$
$f' i x_1 i x_2 = i f' x_1 x_2$

M :
M_f : $f(\cdot)$ $g(\cdot)$ $f'(\cdot,\cdot)$
M_i : $i(\cdot)$
M_a : $a(\cdot)$ $b(\cdot)$ $c(\cdot)$ $d(\cdot)$ $\alpha(\cdot)$ $b'(\cdot)$ $c'(\cdot)$
M_α : α β γ δ α' β' γ'

Figure 1



$A \to B$, $C \to D$
$C \subseteq B$, $A \subseteq D$ (a)



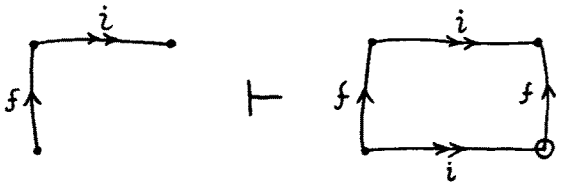
$A_1 \to C_1$, $C_2 \to B_2$, $D_2 \to B_2$

$C_1 D_1 \subseteq C_2 D_2$
$A_3 D_3 \subseteq A_1 D_1$ (b)
$D_3 B_3 \subseteq D_2 B_2$

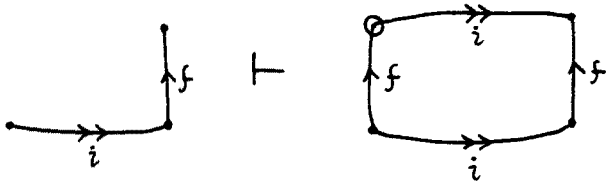
→ red
→→ black

Figure 2

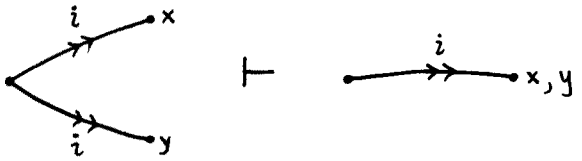
Rule 1



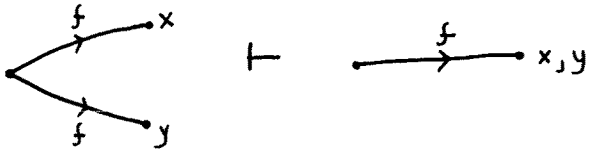
Rule 2



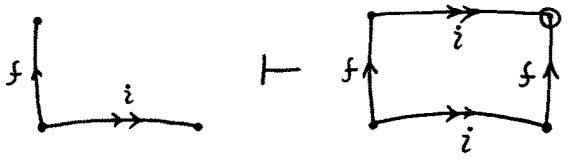
Rule 3



Rule 4



Rule 5
[Mitchell]



⊙ : new node

Figure 3