JOURNAL OF COMPUTER AND SYSTEM SCIENCES 33, 203–233 (1986)

Partition Semantics for Relations

STAVROS S. COSMADAKIS* AND PARIS C. KANELLAKIS†

MIT, Cambridge, Massachusetts

AND

NICOLAS SPYRATOS

Université de Paris Sud, Paris, France

Received May 1, 1986

We use set-theoretic partitions to assign semantics to relation schemes, relations, and dependencies. This approach leads to a natural extension of functional dependencies, the most common database constraints, which is based on the duality between product and sum of partitions. These more general constraints, which we call *partition dependencies* (PDs), have the power to express both functional determination and connectivity in undirected graphs. We show that the implication problem for PDs is the uniform word problem for lattices, and we give a polynomial time algorithm for this implication problem. Our decision procedure for the uniform word problem for lattices improves upon previous exponential decision procedures. The algebraic techniques we use are new to database theory. We investigate the expressive power of PDs, and show that partition semantics justify a number of variants of an assumption commonly used in database theory, namely the weak instance assumption. Finally, we provide a polynomial time test for consistency of a set of relations with a set of PDs. © 1986 Academic Press, Inc.

1. INTRODUCTION

It is customary in database theory to consider the database scheme and database constraints as sentences from predicate calculus, and the relations of the database as interpretations for these sentences. These interpretations either satisfy or falsify the scheme and constraints. An interesting body of theory has developed around this approach, particularly with respect to special types of sentences about relations called dependencies (see [30, 24] for surveys of the area). Here we investigate a different view, first proposed in [29], in which database scheme, constraints, *and* the database are strings of uninterpreted symbols. Interpretations for these symbols are

* Present address: IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y.

† On leave from Brown University; supported partly by NSF Grant MCS-8210830 and partly by ONR-DARPA Grant N00014-83-K-0146, ARPA Order 4786.

0022-0000/86 $3.00

Copyright © 1986 by Academic Press, Inc.
All rights of reproduction in any form reserved.

571/33/2-6

provided using partitions, that is, families of nonempty, disjoint sets whose union is a population of objects. This approach, which we call *partition semantics*, allows us to better understand and generalize such basic database concepts as functional dependencies and weak instances [19, 32].

Partition semantics reveal the algebraic nature of the most common database dependency, the *functional dependency* (FD). Using partition semantics we demonstrate that the implication problem for FDs is a problem of infering an equation between algebraic expressions from other such equations. In more mathematical terms, FD implication is equivalent to *the uniform word problem for idempotent commutative semigroups*. This is a restricted form of *the uniform word problem for lattices* [10, 23]. Our algebraic approach to dependency implication is similar in spirit to the approach of [35], although much closer in its details to standard equational theories [21]. Another (different) use of algebraic methods is made in [9].

Let $\cdot, +$ be two operators satisfying the lattice axioms [10] (see also Sect. 2.2) and let the attributes of the database scheme be considered as constants. Partition semantics allow us, without any loss of generality, to treat an FD, such as $AB \rightarrow CD$ (where $\rightarrow$ is conventional FD notation [30]), as the equation $A \cdot B = A \cdot B \cdot C \cdot D$. Interestingly enough, partition semantics allow us to complete the analogy. An equation $A = B + C$, where $+$ is the natural dual of $\cdot$ in a lattice of partitions, is a dependency expressing a connectivity condition. In general a *partition dependency* (PD) is an equation $e = e'$, where e, e' are algebraic expressions using $\cdot$, $+$, and the attributes. Connectivity cannot be expressed and transitive closure cannot be computed using relational algebra [16, 1]. Since, as we show, PDs cannot express multivalued dependencies (another popular database constraint), the expressive power of PDs is orthogonal to that of other generalizations of FDs [35, 15, 4, 5, 28, 7, 9].

We show that the *implication problem* for PDs can be solved in *polynomial time*. For this we solve efficiently the uniform word problem for lattices. The decidability of the uniform word problem for lattices follows from the finite controlability of a more general class of sentences [14, 13, 25]. Our decision procedure is new and runs in polynomial time. A straightforward implementation of our algorithm takes time $O(n^4)$. In addition to its use in the implication problem, we also use this algorithm to efficiently reduce the problem of *testing consistency of a set of relations with a set of* PDs, to that of testing consistency with a set of FDs [19].

The question of consistency (i.e., whether there exists a partition interpretation satisfying a given database and a given set of PDs), is very much related to the problem of testing consistency of a database and a set of dependencies under the *weak instance assumption*. We demonstrate that these are equivalent problems. The weak instance assumption is a popular assumption used to model incomplete information and to model databases with many relations, together with a set of FDs. The weak instance assumption states that the relations in the database are contained in the projections of a universal relation (i.e., relation over all the attributes), which represents the "real" world.

The exact variant of the weak instance assumption we will be dealing with will depend on whether we assume an *open* or a form of *closed world* interpretation [27]. If we assume a *domain closure axiom* [27], which is formalized in partition semantics by the complete atomic data assumption (CAD) [29], then testing consistency of a database with a set of FDs becomes *NP*-complete. If we assume an open world interpretation, testing consistency of a database with a set of FDs can be done using the efficient weak satisfaction test of [19]. We generalize this test to a set of PDs.

We use the standard relational terminology from [30, 24], as well as the concept of a weak instance for a database and a set of FDs [19, 32]. A useful survey of equational theories can be found in [21]. Computational aspects of these theories are explored in [23, 6, 22].

In order to make this paper self-contained, we provide some necessary definitions in Section 2. Section 2.1 contains useful facts from relational database theory and Section 2.2 from lattice theory [10].

Partition semantics are the subject of Section 3. The formal definitions of partition semantics (in particular the notions of partition interpretation and satisfaction) are in Section 3.1. The definition of partition dependencies and examples, illustrating functional determination and connectivity, follow in Section 3.2. This section also explains other possible restrictions on interpretations, such as CAD.

The expressive power of partition dependencies is the topic of Section 4. The correspondence between FDs and a restricted form of PDs, called *functional* partition dependencies (FPDs), is investigated in Section 4.1. We show that PDs can express FDs without loss of generality; this is accomplished using a canonical partition interpretation. In Section 4.2 we show, using undirected graph connectivity, that the expressive power of PDs differs from that of first-order sentences. This result is similar to results in [16, 1] and holds even if we restrict attention to finite interpretations. We also show that PDs cannot express multivalued dependencies. In Section 4.3 we demonstrate the connection between consistency under the weak instance assumption and consistency of a set of relations and a set of PDs.

A solution to the PD impliation problem is contained in Section 5. We first present, in Section 5.1, a new algebraic view of FD implication and PD implication as uniform word problems in lattices. The decidability of this word problem follows from [25, 14, 13]. The decision procedure of [25] requires exponential time. A new polynomial-time inference algorithm for PDs and the uniform word problem for lattices is given in Section 5.2. The proofs in these sections use new techniques for database theory, which derive from universal algebra and the theory of lattices [10, 23, 18]. The uniform word problem for lattices is complete in polynomial time. This distinguishes it from the problem of recognizing lattice identities (or equivalently recognizing the PDs which are always true), which is solvable in logarithmic space, as we show in Section 5.3. This logspace bound improves upon polynomial time results in [22, 6]. In Section 5.3 we also compare our lattice theoretic analysis to the database theory of FDs.

In Section 6 we investigate the complexity of testing a database and database

constraints for consistency, i.e., testing whether there is a partition interpretation satisfying these syntactic objects. If we assume complete atomic data and FDs our task in *NP*-complete; Section 6.1. However, if we do not assume complete atomic data (Sect. 6.2), then there is a polynomial time test for consistency of a set of relations with a set of PDs. This test combines the test of [19] with the efficient inference algorithm for PDs. Section 7 contains conclusions and open questions.

The material in Section 3 is the contribution of the third author. The material in Sections 4, 5, 6 is the contribution of the first two authors.

2. Basic Concepts

2.1. *Relations*

Let $\mathcal{U}$ be a finite set of *attributes* $\{A, B, C,..., A_1, B_1, C_1,..., Q\}$, and $\mathcal{D}$ a countably infinite set of *symbols* $\{a, b, c,..., a_1, b_1, c_1,...\}$, such that $\mathcal{U} \cap \mathcal{D} = \emptyset$. A *relation scheme* is an object $R[U]$, where R is the *name* of the relation scheme and $U \subseteq \mathcal{U}$. A *tuple* t over U is a function from U to $\mathcal{D}$. If $t[A_i] = a_i$, where $U = \{A_1,..., A_k\}$, then we denote tuple t as $a_1 a_2,..., a_k$, and the restriction of t on a subset X of U as $t[X]$. A *relation* r over U is a set of tuples over U. The restriction of r on a subset X of U is called the *projection* of r on X and denoted as $r[X]$. We assume that, in general, there are both finite and infinite relations. A *database scheme* is a finite set of relation schemes $D = \{R_1[U_1],..., R_m[U_m]\}$ and a *database* $d = \{r_1,..., r_m\}$ associates each relation scheme $R_i[U_i]$ in D with a relation r_i over U_i.

A database is represented, in the natural fashion, by a set of tables with the relation schemes as headers and the tuples as rows. Each column is headed by an attribute. We use $d[A]$ to denote the set of symbols appearing in database d under *all* columns headed by attribute A.

Let d be a database over database scheme D (as above). Let the union of all attributes in D be U. A relation w, over U, is a *weak instance* for d iff every tuple of relation r_i (over U_i) of d appears in $w[U_i]$. In other words, the *projection* of w on U_i must contain r_i.

Assume $R[U]$ is a relation scheme and X, Y are nonempty subsets of U. We call $R: X \to Y$ a *functional dependency* (FD). A relation r, with relation scheme $R[U]$, satisfies $R: X \to Y$ iff: whenever t, h are tuples in r with $t[X] = h[X]$, then $t[Y] = h[Y]$. We use $X \to Y$ if $R[U]$ is clear from the context. FDs have a number of interesting computational properties (e.g., [2, 3, 19]).

Let d be a given database over database scheme D. Let the union of all attributes in D be the set U and let Σ be a given set of FDs on relation scheme $R[U]$. We say that d is consistent with Σ under the weak instance assumption iff there exists a weak instance w for d which satisfies the set of FDs Σ. An efficient test for consistency is contained in [19].

Functional dependencies are database constraints expressed using sentences of

first-order predicate calculus with equality. This is also true for other dependencies investigated in the literature [35, 15, 4, 5, 28, 7]. In this framework dependencies are syntactic objects, and the database is a semantic object satisfying or falsifying the dependencies. Let r be a relation, Σ a set of dependencies, and σ a dependency. We denote satisfaction of σ by r as $r \models \sigma$. We also say that Σ (finitely) implies σ, if every (finite) relation which satisfies the dependencies in Σ, also satisfies the dependency σ. We denote implication and finite implication as $\Sigma \models_{\text{rel}} \sigma$ and $\Sigma \models_{\text{rel,fin}} \sigma$, respectively. We use the subscript $_{\text{rel}}$ to stress that this is implication over relations.

2.2. *Lattices*

Let L be a nonempty set with two binary operations $\cdot$, $+$. Set L is a *lattice* [10] if the following axioms hold for all its elements x, y, z:

LA Axioms

associativity: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$, $(x + y) + z = x + (y + z)$

commutativity: $x \cdot y = y \cdot x$, $x + y = y + x$

idempotence: $x \cdot x = x$, $x + x = x$

absorption: $x + (x \cdot y) = x$, $x \cdot (x + y) = x$.

Let us define a relation $x \leqslant y$ between elements x, y of lattice L such that $x \leqslant y$ iff $x = x \cdot y$ iff $y = y + x$. This relation is a *partial order*, the natural partial order of the lattice.

A *sublattice* of lattice L is a subset of L closed under the $\cdot$ and $+$ operations.

Consider the set of attributes $\mathcal{U}$ of Section 2.1, and let U be a subset of $\mathcal{U}$. We will say that L is a *lattice with constants over* U if we have a function g mapping each element of U to an element of the lattice, (i.e., if for each attribute A in U there is exactly one element in L named A). Note that an element in L could have more than one name. When U and g are obvious from the context we will refer to lattice L, with constants over U, as lattice L.

Let $W(U)$ denote the set of finite expressions built using the elements of U as generators and $\cdot$, $+$ as binary operators, (i.e., these are uninterpreted strings formed using the symbols in $U, \cdot, +, (,)$ and representing *ground terms* [23]). If L is a lattice with constants over U, each of these expressions can be interpreted (in the obvious fashion) as a name for exactly one element in L. Let e and e' be expressions in $W(U)$. We will say that lattice L with constants over U satisfies $e = e'$ if the elements in L with names e and e' are identical. We denote this satisfaction by $L \models e = e'$.

For δ an equation $e = e'$ as above, and E a set of such equations, we say that $E \cup LA$ implies δ, or $E \models_{\text{lat}} \delta$, if every lattice L with constants over U which satisfies each equation in E also satisfies δ. We use the subscript $_{\text{lat}}$ to stress that this is implication over lattices. This implication problem is known as the *uniform word problem for lattices*. If we restrict attention to only finite lattices we have finite implication, $E \models_{\text{lat,fin}} \delta$.

3. Partition Semantics

3.1. *Partition Interpretations*

The syntactic objects of "conventional" database theory are relation schemes, database schemes, and database constraints. To all of these syntactic objects we assign partition semantics. We also assign partition semantics to the symbols in $\mathscr{D}$, tuples, relations, and databases. The interpretations of these objects are defined starting from an interpretation for the attributes, which we call a partition interpretation.

Definition 1. A *partition interpretation* $\mathscr{I}$, over attributes $\mathscr{U}$, is a set of triples $\{(\rho_A, \pi_A, f_A) \mid A \in \mathscr{U}\}$, where for each attribute A:

1. ρ_A is a nonempty set, the *population* of A,
2. π_A is a partition of ρ_A, the *atomic partition* of A,
3. f_A is a function from $\mathscr{D}$ to $\pi_A \cup \{\varnothing\}$, such that,

$$\pi_A = \{f_A(x) \mid x \in \mathscr{D}, f_A(x) \neq \varnothing\} \quad \text{and} \quad \text{if } x \neq y \text{ then } f_A(x) \cap f_A(y) = \varnothing.$$

Note that in the above definition π_A is a partition in the mathematical sense, i.e., a family of nonempty, disjoint subsets of ρ_A whose union is ρ_A. Also f_A maps a unique symbol from $\mathscr{D}$ to each distinct member of π_A, and maps the rest of $\mathscr{D}$ to $\varnothing$.

There are two natural operations on partitions. Given a partition π of set (or population) ρ and a partition π' of set (or population) ρ', their *product* $\cdot$ is defined as

$$\pi \cdot \pi' = \{x \mid x = y \cap z \neq \varnothing, y \in \pi, z \in \pi\}.$$

Their *sum* $+$ is defined as

$$\pi + \pi' = \{x \mid \text{two elements } \mu, \nu \text{ in } \rho \cup \rho' \text{ are in set } x \text{ if and only if there is a chain of sets } x_1, x_2, \ldots, x_q \text{ from } \pi \cup \pi' \text{ such that } \mu \in x_1, \nu \in x_q, \text{ and } x_i \cap x_{i+1} \neq \varnothing \text{ for all } i,\ 1 \leqslant i \leqslant q-1\}.$$

Note that $\pi \cdot \pi'$ is a partition of the population $\rho \cap \rho'$ and $\pi + \pi'$ is a partition of the population $\rho \cup \rho'$. It is easy to verify that both $\cdot$, $+$ are *associative*, *commutative*, and *idempotent*. It is standard terminology to refer to members of partitions as *blocks*. Also if $\rho = \rho'$ then $\cdot$ produces the *coarsest common refinement* of π and π' and $+$ produces their *finest common generalization*. The choice of letters $\cdot$ and $+$ is not arbitrary, since as we shall see below, they are lattice operations.

The attributes in $\mathscr{U}$ and the *uninterpreted operator symbols* $\cdot$, $+$ can be used to build finite expressions. We call such expressions *partition expressions*, over $\mathscr{U}$: every attribute is a partition expression and if e, e' are partition expressions then so are $(e \cdot e')$ and $(e + e')$. We use $W(\mathscr{U})$ for the set of partition expressions. When $\mathscr{U}$ is

obvious from the context we refer to partition expressions over $\mathscr{U}$ and partition interpretations over $\mathscr{U}$ simply as partition expressions and partition interpretations.

Given a partition interpretation $\mathscr{I}$, the semantics (*meaning*) of a partition expression is defined using structural induction and interpreting the $\cdot$, $+$ symbols as partition product and sum, respectively. If partition expressions e, e', in partition interpretation $\mathscr{I}$, have meanings the partitions π, π' of populations ρ, ρ' respectively, then:

1. The meaning of attribute A is partition π_A of population ρ_A.
2. The meaning of $(e \cdot e')$ is partition $\pi \cdot \pi'$ of population $\rho \cap \rho'$.
3. The meaning of $(e + e')$ is partition $\pi + \pi'$ of population $\rho \cup \rho'$.

Given a partition interpretation $\mathscr{I}$, the meaning of a relation scheme $R[U]$, $U = \{A_1, A_2, ..., A_n\}$, is defined to be the same as the meaning of the partition expression $A_1 \cdot A_2 \cdot \cdots \cdot A_n$. This meaning is well defined because of the associativity, commutativity, and idempotence of $\cdot$. For example, the meaning of $R[ABC]$ in $\mathscr{I}$ is the (*composite*) partition $\pi_A \cdot \pi_B \cdot \pi_C$. Note that a relation scheme $R_1[ABC]$, with a different relation name but over the same attributes, has the same meaning as $R[ABC]$. Thus attributes play the critical role in determining partition semantics for many relation schemes.

A partition interpretation $\mathscr{I}$ can also be used to define the meaning of a tuple t in relation r whose scheme is $R[U]$. The meaning of symbol $t[A]$ (the symbol of t in the A column) is $f_A(t[A])$. This meaning is either $\varnothing$ or a block of the atomic partition π_A. The meaning of tuple t is $\bigcap_{A \in U} f_A(t[A])$. This meaning is either $\varnothing$ or a block of the composite partition, which is the semantics of $R[U]$. For example, let tuple xyz be in a relation of database d whose scheme is $R[ABC]$. Also suppose that in $\mathscr{I}$ we have $f_A(x) = a$, $f_B(y) = b$, $f_C(z) = c$, where a, b, c are blocks from partitions π_A, π_B, π_C, respectively. Then the meaning of $t[A]$ in $\mathscr{I}$ is a and the meaning of t is $a \cap b \cap c$. In a different partiton interpretation $\mathscr{I}'$, which is identical with $\mathscr{I}$ with the exception that $f_A(x) = \varnothing$, the meaning of $t[A]$ is $\varnothing$ and that of t is also $\varnothing$.

Starting from a partition interpretation $\mathscr{I}$ we have assigned partition semantics to attributes, partition expressions, relation schemes, database schemes. For this we have used the population and atomic partition parts of $\mathscr{I}$. Using also the (naming) functions f_A we then assigned sematics to symbols in tuples, relations, and databases. This strict separation of semantics from syntax allows us to define when a partition interpretation satisfies a given database.

DEFINITION 2. Let d be a database and $\mathscr{I}$ a partition interpretation. $\mathscr{I}$ satisfies d (notation: $\mathscr{I} \models d$), iff:

$$\forall r \in d, \forall t \in r. \bigcap_{A \in U} f_A(t[A]) \neq \varnothing, \text{ where } R[U] \text{ is the relation scheme of relation } r.$$

Intuitively a partition interpretation $\mathscr{I}$ satisfies database d if every tuple in every

relation of d represents a nonempty set (i.e., there is a set of "objects" corresponding to this tuple). Note that it is possible to have nonempty sets defined by the underlying populations which do not correspond to any tuple in d.

3.2. *Partition Dependencies*

In this section we explain various means of expressing restrictions on partition interpretations, other than the restriction of satisfying database d.

DEFINITION 3. A *partition dependency* (PD) is an equation $e = e'$, where e, e' are partition expressions. A partition interpretation $\mathscr{I}$ satisfies $e = e'$ (notation: $\mathscr{I} \models e = e'$), iff:

> $\pi = \pi'$ and $\rho = \rho'$, where partitions π, π' of populations ρ, ρ' are the respective meanings of e, e' in $\mathscr{I}$.

Using elementary properties of partitions one can easily derive some important properties of PDs. Let x, y, z be any partition expressions over $\mathscr{U}$. The following PDs are true in any partition interpretation over $\mathscr{U}$:

associativity: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$, $(x + y) + z = x + (y + z)$

commutativity: $x \cdot y = y \cdot x$, $x + y = y + x$

idempotence: $x \cdot x = x$, $x + x = x$

absorption: $x + (x \cdot y) = x$, $x \cdot (x + y) = x$.

The above properties show that, given a partition interpretation $\mathscr{I}$ over $\mathscr{U}$, the set of partitions produced by closing $\mathscr{I}$'s atomic partitions under product and sum is a *lattice*. Every A in $\mathscr{U}$ is mapped on the atomic partition of A, which is an element of this lattice, therefore we have a lattice with constants over $\mathscr{U}$. We denote this lattice as $L(\mathscr{I})$. Each attribute A in $\mathscr{U}$ will be the name of an element of $L(\mathscr{I})$, i.e., of partition π_A. Thus,

THEOREM 1. *Let $\mathscr{I}$ be a partition interpretation over $\mathscr{U}$ and $e = e'$ be a PD. The set of partitions $L(\mathscr{I})$ produced by closing $\mathscr{I}$'s atomic partitions under product and sum is a lattice, with constants over $\mathscr{U}$. Also, $\mathscr{I} \models e = e'$ iff $L(\mathscr{I}) \models e = e'$.*

Let U be a set of attributes $\{A_1, \ldots, A_n\}$. With a slight abuse of notation, in the context of a PD we use U for the partition expression $A_1 \cdot \cdots \cdot A_n$. This is consistent with the definition of the meaning of a relation scheme. We can now present a special important class of partition dependencies. Let X, Y be nonempty sets of attributes. A *functional partition dependency* (FPD) is a partition dependency of the form $X = X \cdot Y$. It follows from Definition 3 above and a simple induction that:

THEOREM 2. *If $\mathscr{I}$ is a partition interpretation and if partitions π, π' of populations ρ, ρ' are the meanings of sets of attributes X, Y in $\mathscr{I}$, then $\mathscr{I} \models X = X \cdot Y$ iff*

1. $\forall x \in \pi, \exists y \in \pi' \quad x \subseteq y$
2. $\rho \subseteq \rho'$.

Because of the duality of $\cdot$ and of $+$ in the lattice $L(\mathscr{I})$ [10] the FPD $X = X \cdot Y$ is equivalent to the PD $Y = Y + X$. For this FPD we can use the notation $X \leqslant Y$, because the $\leqslant$ may be interpreted using the natural partial order (Sect. 2.2) of the lattice $L(\mathscr{I})$. Thus, for X, Y nonempty sets of attributes from $\mathscr{U}$ we have encountered three ways of expressing an FPD: $X = X \cdot Y$, $Y = Y + X$, or $X \leqslant Y$. We will see that these are very much related to the FD $X \to Y$, in a relation scheme with attributes $\mathscr{U}$.

Definitions 1, 2 (Sect. 3.1) and Definition 3 above, as well as Theorems 1 and 2 above are illustrated in Fig. 1. Observe from Fig. 1 that $L(\mathscr{I})$ need not be *distributive* [10].

EXAMPLE a. Assume we have two attributes A for employee-number and B for manager-number and we wish to express the fact that: each employee can be associated with only one manager. This is analogous to the familiar functional

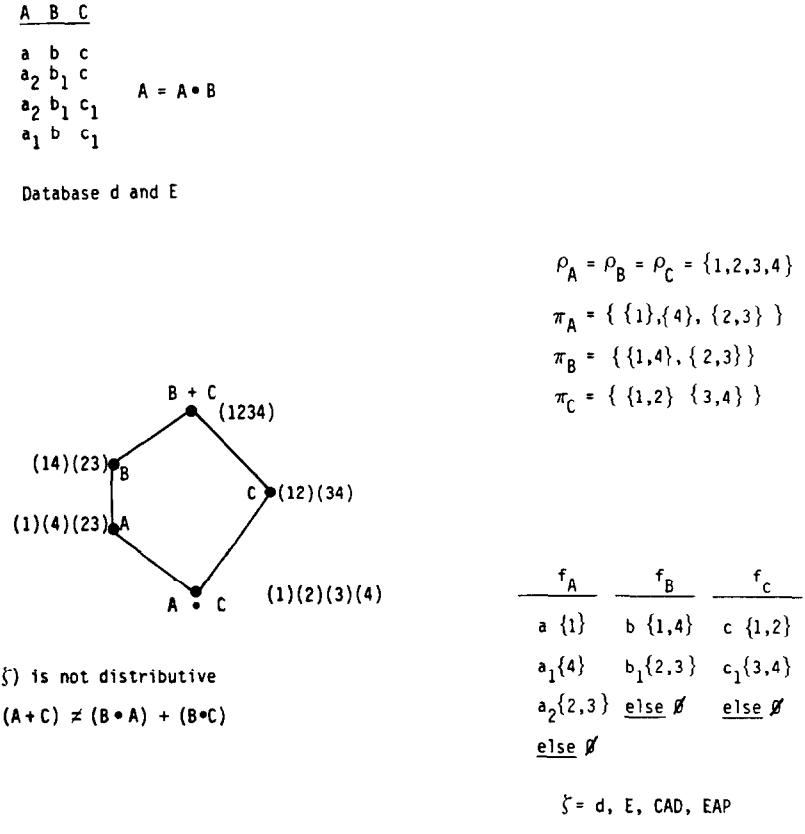

FIGURE 1

dependency for relations $A \to B$. We express this constraint as the FPD $A = A \cdot B$. Intuitively the above FPD guarantees that two individuals of the population of A in the same block of π_A (e.g., all unlucky individuals whose employee number is 13) are also in the same block of π_B (e.g., the set of individuals whose manager has manager number 7). A fine point is that in each interpretation satisfying this FPD we have $\rho_A \subseteq \rho_B$. Therefore, by Theorem 2, B is uniquely determined for any possible A. Note that the semantics in this case allows interpretations in which a manager manages individuals who do not have employee numbers, although every individual with an employee number must be managed by a manager. An equivalent way to express this functional determination is to use the PD $A + B = B$ or $A \leqslant B$.

EXAMPLE b. PDs have certain similarities with ISA relationships such as: every car (attribute C) is a vehicle (attribute B). A surprising fact about partition interpretations is that distinctions between expressing functional determination and this type of relationship seem to disappear (at first sight). This is because ISA relationships implicitly define a function from subset to superset. In this example the natural FPD constraint is: $C = C \cdot B$. In any satisfying interpretation of this FPD we have $\rho_C \subseteq \rho_B$, and a car block functionally determines a vehicle block. The precise expression of ISA relationships using PDs and other statements is still a matter or research [29]. We would like to stress here that the treatment of ISA relationships in conventional database theory is quite different. *Inclusion dependencies* [7] are the database constraints used. The algebraic properties of inclusion dependencies are very different from the properties of PDs [9, 8].

EXAMPLE c. Let ρ be a population of cars and ρ' a population of bicycles, and let $\rho \cap \rho' = \varnothing$. We wish to express that: every vehicle is either a car or a bicyle. Let C, car registration number, be interpreted as a partition π of ρ, and B, bicycle registration number, be interpreted as a partition π' of ρ'; intuitively the database cannot distinguish between cars with the same car registration or between bicycles with the same bicycle registration. Since the two populations are distinct $\pi + \pi' = \pi \cup \pi'$, that is $+$ produces the union of two families of blocks. In this case one may use the PD: $A = C + B$, where A is vehicle registration number and C and B have disjoint populations (as an additional restriction).

EXAMPLE d. Suppose we wish to keep a relation of cars C, which are complex objects with characteristics registration number A and factory serial number B. We can express it as a composite partition using the semantics of relation schemes: $C = A \cdot B$.

EXAMPLE e. Consider a database d with only one relation r representing an undirected graph. This relation has three attributes: A head, B tail, and C component. For every edge $\{a, b\}$ in the graph we have in the relation tuples *abc*, *bac*,

aac, *bbc*, where *c* is a number with could vary with *a* and *b*. These are the only tuples in *r*. Note that by the way *r* was constructed from the graph we may, without loss of generality, restrict our attention to interpretations satisfying *d* with $\rho_A = \rho_B$. We would like to express that: *C* is the connected component in which the arc (*A*, *B*) belongs. We can do this by restricting partition interpretations to: $\mathscr{I} \models (d$ and $C = A + B)$. This last example illustrates how by regarding both *d* and a PD as syntactic objects we can express "essential" aspects of undirected connectivity, (see Theorem 4, Sect. 4).

As some of the above examples indicate, there are other natural restrictions that can be imposed on partition interpretations.

DEFINITION 4. Assuming $\mathscr{I}$ satisfies database *d* and a set of PDs, we say that:

1. $\mathscr{I}$ satisfies the *complete atomic data assumption* (CAD) iff:

$$\forall A \in \mathscr{U}, \forall x \in \mathscr{D} \qquad (x \in d[A] \text{ iff } f_A(x) \neq \varnothing).$$

2. $\mathscr{I}$ satisfies the *equal atomic populations assumption* (EAP) iff $\forall A, B \in \mathscr{U}$. $\rho_A = \rho_B$.

The CAD assumption is analogous to a domain closure axiom [27]. It says that the only true atomic facts about attribute *A* are the ones in database *d* in the columns headed by *A*. As in Definition 2 (Sect. 3.1), true is equivalent to having nonempty meaning. The EAP assumption restricts attention to attributes with common populations. An interesting fact is that, in contrast to CAD which turns out to be quite restrictive from a complexity point of view, EAP has no significant effect on our analysis. Another (additional) assumption which would allow us to use + in order to express $\cup$, as in Example c above, would be that: "two attributes have disjoint populations."

4. EXPRESSIVE POWER

In Sections 4.1 and 4.2 we will restrict attention to databases consisting of a single relation *r*. We want to study what kinds of things can be said about *r* by sets of PDs. To do this, however, we first have to define what it *means* for *r* to satisfy a PD δ, or $r \models \delta$. This will complement our definitions of satisfaction of a "conventional" dependency by a relation (Sect. 2.1), and of satisfaction of a PD by a lattice and a partition interpretation (Sect. 2.2 and 3.2). Relations will then provide the common ground for comparison of PDs with other dependencies.

In Section 4.3 we will study the case of many relation schemes. The emphasis of partition semantics on attributes is the reason for the strong relationship between partition interpretations and weak instances.

4.1. *Relations and Partitions*

Given a relation r we can define a *canonical* partition interpretation $I(r)$ corresponding to r.

DEFINITION 5. Let r be a relation over $\mathcal{U}$; $I(r)$ is the following partition interpretation over $\mathcal{U}$:

1. for each A in $\mathcal{U}$, $\rho_A = \{i_t \mid t \text{ is a tuple of } r\}$ (if t is tuple of r, let i_t be a positive integer unique to t);
2. for each x in $\mathcal{D}$, $f_A(x) = \{i_t \mid t[A] = x\}$;
3. for each A in $\mathcal{U}$, π_A is the partition induced by f_A on ρ_A.

Note that $I(r)$ satisfies EAP. Also $I(r) \models r$ for any relation r. We also define, given a partition interpretation $\mathcal{I}$, a *canonical* relation $\mathcal{R}(\mathcal{I})$ that corresponds to it.

DEFINITION 6. Let $\mathcal{I}$ be a partition interpretation over $\mathcal{U}$, and ρ the union of its populations $\bigcup_{A \in \mathcal{U}} \rho_A$. $\mathcal{R}(\mathcal{I})$ is a relation. For each i in ρ there is a tuple t_i in $\mathcal{R}(\mathcal{I})$ such that: $t_i[A] = x$ if $i \in f_A(x)$, and $t_i[A] = i_A$ if $t \notin \rho_A$; (i_A is a symbol unique to i and A in the relation constructed).

Observe that because $I(r)$ satisfies EAP one can easily argue that $\mathcal{R}(I(r)) = r$. But $I(\mathcal{R}(\mathcal{I}))$ is not necessarily $\mathcal{I}$, because $\mathcal{I}$ is not required to satisfy the EAP restriction. If EAP holds in $\mathcal{I}$ then $I(\mathcal{R}(\mathcal{I}))$ is very similar to $\mathcal{I}$. The only possible difference is if two elements i, j of ρ (the common population) are in the same blocks for all atomic partitions of $\mathcal{I}$, then t_i and t_j are copies of the same tuple in $\mathcal{R}(\mathcal{I})$, thus $I(\mathcal{R}(\mathcal{I}))$ contains only one of elements i, j (by Definition 6 and 7). One can argue however that: if EAP holds in $\mathcal{I}$ then $L(I(\mathcal{R}(\mathcal{I}))) = L(\mathcal{I})$.

THEOREM 3. a. *If* $\mathcal{I} \models X = X \cdot Y$ *then* $\mathcal{R}(\mathcal{I}) \models X \to Y$.

b. $r \models X \to Y$ *iff* $I(r) \models X = X \cdot Y$.

Proof. a. Let t_i, t_j be distinct tuples of $\mathcal{R}(\mathcal{I})$, such that $t_i[X] = t_j[X]$. Then for each A in X, $t_i[A] = t_j[A]$, which means $i, j \in a$ for some $a \in \pi_A$ (where $a = f_A(x)$ for some x). But $\mathcal{I} \models X = X \cdot Y$, and thus for each B in Y, $i, j \in b$ for some $b \in \pi_B$, i.e., $t_i[B] = t_j[B]$. Thus, $\mathcal{R}(\mathcal{I}) \models X \to Y$.

b. The "if" direction follows from part (a) of this proposition, by observing that $\mathcal{R}(I(r)) = r$. The argument for the "only if" direction is as follows: Let i_t, i_h be elements (of the population over which $I(r)$ is defined) such that, for each A in X, $i_t, i_h \in a$ for some $a \in \pi_A$. This means $t[X] = h[X]$, and since $r \models X \to Y$, $t[B] = h[B]$ for each B in Y. But then for each B in Y, $i_t, i_h \in b$ for some $b \in \pi_B$. and thus $I(r) \models X = X \cdot Y$ (recall Theorem 2, Sect. 3.2). ∎

If $\mathcal{I}$ satisfied EAP then the converse of Theorem 3a would hold; in general, however, it does not. Theorem 3a supports the claim (see Ex. a, Sect. 3.2) that the

FPD $X = X \cdot Y$ is the correct counterpart of the FD $X \to Y$. It motivates our final definition.

DEFINITION 7. A relation r satisfies a PD δ (notation: $r \models \delta$) iff $I(r) \models \delta$.

Given this definition, one easily sees the following for attributes A, B, C:

(I) $r \models C = A \cdot B$ iff for any tuples $t, h \in r$, $t[C] = h[C]$ iff ($t[A] = h[A]$ and $t[B] = h[B]$).

(II) $r \models C = A + B$ iff for any tuples $t, h \in r$, $t[C] = h[C]$ iff ($\exists n \geqslant 0$, $t_0, \ldots, t_n$ tuples of r: $t = t_0$, $t_n = h$, and for $i = 0, \ldots, n-1$, $t_i[A] = t_{i+1}[A]$ or $t_i[B] = t_{i+1}[B]$).

(III) $r \models C = A \cdot B$ iff for any tuples $t, h \in r$, $t[C] = [C]$ iff ($\exists n \geqslant 0$, $t_0, \ldots, t_n$ tuples of r: $t = t_0$, $t_n = h$, and for $i = 0, \ldots, n-1$, $t_i[A] = t_{i+1}[A]$ and $t_i[B] = t_{i+1}[B]$).

We have now reduced our interpretations to relations, in order to be able to compare PDs with other dependencies. It becomes evident from Theorem 3b and Definition 7 that $r \models X = X \cdot Y$ iff $r \models X \to Y$. From (II) it is clear that the relation of Example e (Sect. 3.2) correctly represents the connected components of an undirected graph iff it satisfies $C = A + B$. Finally, (III) right above is trivially equivalent to (I), and it illustrates an and/or duality between $\cdot/+$.

4.2. *Expressive Power of PDs*

We now want to compare the expressive power of PDs to that of previously studied database constraints. The majority of these database constraints can be described by sets of sentences of first-order predicate calculus with equality and one relation symbol R.

We will say that a database dependency σ *is expressed* by a set E of PDs when for any relation r, $r \models \sigma$ iff $r \models E$. Also we will say that a PD δ *is expressed* by a set of sentences Σ when for any relation r, $r \models \delta$ iff $r \models \Sigma$. If we restrict the above definitions to finite relations r, then we have *expressibility over finite relations* [16, 1].

EXAMPLE f. Let X, Y be sets of attributes. From the algebraic properties of $\cdot$, the PD δ: $X = Y \cdot Z$ is equivalent to $X = X \cdot Y \cdot Z$ **and** $Y \cdot Z = X \cdot Y \cdot Z$. For one direction of the equivalence note that the right-hand sides in these two equalities are the same strings. For the other direction use the idempotence of $\cdot$. Therefore, δ is expressed by the set $\{X \to YZ,\ YZ \to X\}$.

Because of Example e (Sect. 3.2), it should come as no surprise [16, 1] that the PD $C = A + B$ cannot be expressed by any set of first-order sentences with a single ternary relation symbol. The following theorem is based on a direct compactness argument. The theorem also holds for expressibility over finite relations (in that case the proof follows directly from [16, 1]).

THEOREM 4. *Let $\mathcal{U} = ABC$; the PD $C = A + B$ cannot be expressed by any set of first-order sentences, with a single ternary relation symbol R as the only non-logical symbol.*

Proof. Our proof will be by contradiction. Let Σ be a set of first-order sentences (with a single ternary relation symbol R as the only non-logical symbol), which expresses $C = A + B$.

For each $k \geqslant 1$, let φ_k be the following first-order formula, with free variables t, h: "$t[C] = h[C]$ and there is *no* sequence $t_0,\ldots, t_k$ such that $t = t_0$, $t_k = h$, and for $j = 0,\ldots, k-1$, $t_j[A] = t_{j+1}[A]$ or $t_j[B] = t_{j+1}[B]$." (One can easily write each φ_k as a first-order formula with R as the only non-logical symbol.)

Now let i be even and consider the relation $r_i = \{1.2.0, 3.2.0, 3.4.0, 5.4.0,\ldots, i-1.i.0, i+1.i.0, i+1.i+2.0\}$ ($a.b.c$ is the tuple with a in the A, b in the B, and c in the C column). Let tuple $t_i = 1.2.0$ and tuple $h_i = i+1.i+2.0$.

Observe that relation r_i and tuples t_i, h_i form a satisfying interpretation for $\Sigma \cup \{\varphi_k, k \leqslant i-1\}$, because $r_i \models C = A + B$ so $r_i \models \Sigma$, and clearly $r_i \models \varphi_k$, $k \leqslant i-1$ (the only possible sequence of tuples is of length i). Thus, any finite subset of $\Sigma' = \Sigma \cup \{\varphi_k : k \geqslant 1\}$ has a satisfying interpretation, and thus by the compactness theorem [12] Σ' has one as well. Let us call this interpretation r', and the two of its distinct tuples interpreting the two free variables tuples t', h'. But this is a contradiction; since r' satisfies Σ, r' satisfies $C = A + B$. On the other hand, this interpretation satisfies φ_k *for all* $k \geqslant 1$, so the tuples t', h' have equal C entries but violate (II) in Section 4.1 (the equivalent way of describing satisfaction of $C = A + B$); thus, r' does not satisfy $C = A + B$. ∎

The above proof can actually be used to show a slightly stronger statement. From the lattice axioms, by trivial algebraic manipulation $C = A + B$ is equivalent to $C = C \cdot (A + B)$ and $A + B = (A + B) \cdot C$. As we will see from the identities in a lattice (Sect. 5), $A + B = (A + B) \cdot C$ is equivalent to $A = A \cdot C$ and $B = B \cdot C$, both FPDs. And if we use the lattice partial order notation, $C = C \cdot (A + B)$ is equivalent to $C \leqslant A + B$. The proof of Theorem 4 above works for the PD $C \leqslant A + B$. The non-first order character of $C \leqslant A + B$ indicates why PDs are a nontrivial extension of FDs:

$$r \models C \leqslant A + B \text{ iff for any tuples } t, h \in r,$$

$$t[C] = h[C] \text{ implies } (\exists n \geqslant 0 \text{ and } t_0,\ldots, t_n \text{ tuples of } r\text{: } t = t_0,\ t_n = h,$$
$$\text{and for } i = 0,\ldots, n-1,\ t_i[A] = t_{i+1}[A] \text{ or } t_i[B] = t_{i+1}[B]).$$

Unfortunately, a dependency as simple as the simplest multivalued dependency (MVD), cannot be expressed by PDs. Let φ be the following multivalued dependency, in predicate logic notation:

$$\varphi = \forall xyzuv.\,[R(xyu) \wedge R(xvz)] \Rightarrow R(xyz).$$

THEOREM 5. *Let $\mathcal{U} = ABC$; the MVD φ cannot by expressed by any set of PDs.*

Proof. Let E be a set of PDs which expresses φ. Referring to Fig. 2, relation r_1 satisfies φ, so $I(r_1) \models E$, and $L(I(r_1))$ (the lattice obtained from $I(r_1)$ by closing under sums and products) satisfies E (Theorem 1, Sect. 3.2). On the other hand, r_2 does not satisfy φ, so $I(r_2)$ does not satify E, and again by Theorem 1, $L(I(r_2))$ does not satisfy E. But this is a contradiction, because $L(I(r_1))$, $L(I(r_2))$ are *isomorphic*, and thus they must satisfy exactly the same PDs. ∎

The proof of Theorem 5 extends with no modification to expressibility over finite relations.

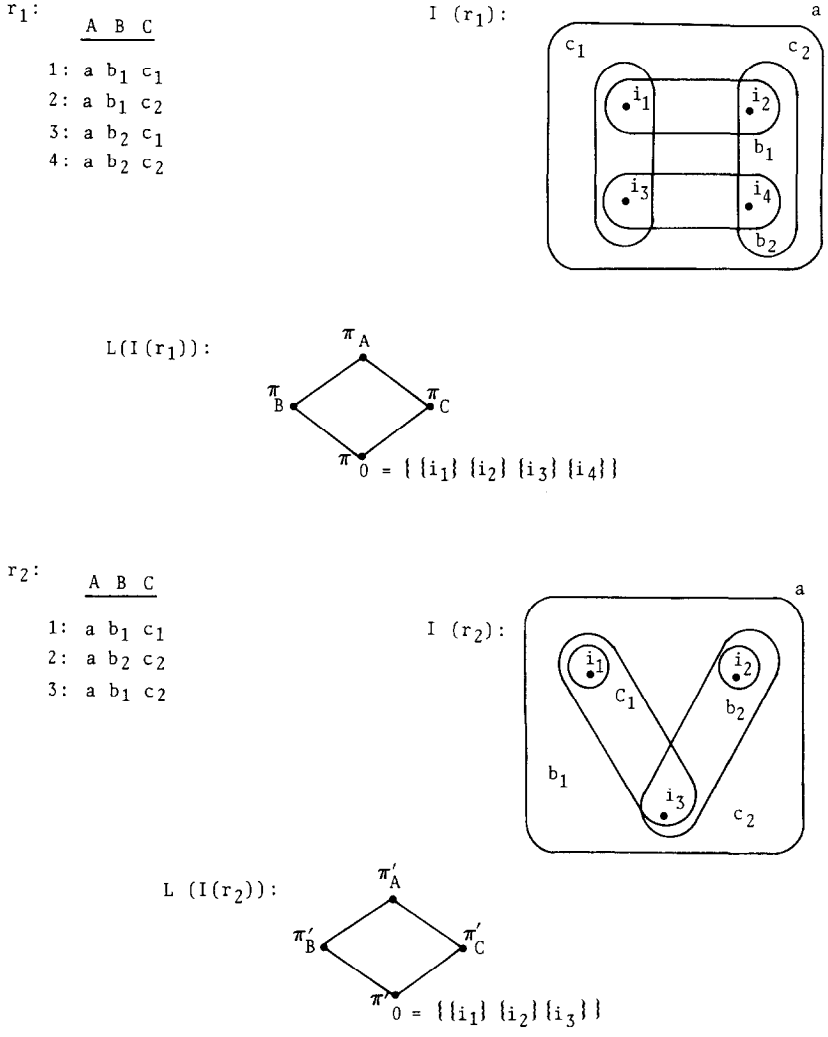

FIGURE 2

4.3. *Partitions and Weak Instances*

Given a database d such that the union of all its attributes is $\mathcal{U}$, and a set E of FPDs over $\mathcal{U}$, let $E_F = \{X \to Y \mid X = X \cdot Y \text{ is in } E\}$. We ask the following two natural questions:

Is there a partition interpretation $\mathcal{I}$ such that $\mathcal{I} \models d, E$?

Is there a partition interpretation $\mathcal{I}$ such that $\mathcal{I} \models d, E$ and $\mathcal{I}$ satisfies CAD and EAP?

Recall that a relation w is a *weak instance* for a database d iff every tuple of relation r (of schema $R_i[U_i]$) of d appears in the projection of w on U_i [19, 32].

THEOREM 6. *Let d be a database and E a set of FPDs:*

a. *There is an $\mathcal{I}$ such that $\mathcal{I} \models d, E$ iff there is a weak instance for d satisfying E_F.*

b. *There is an $\mathcal{I}$ such that $\mathcal{I} \models d, E$ and $\mathcal{I} \models CAD, EAP$ iff there is a weak instance w for d which satisfies E_F and $w[A] = d[A]$, for A in $\mathcal{U}$.*

Proof. a. ($\Rightarrow$) Suppose $\mathcal{I} \models d, E$ and let w be $\mathcal{R}(\mathcal{I})$ (Definition 6, Sect. 4.1). It is easy to see that w is a weak instance for d: if xyz is a tuple appearing in relation $R[ABC]$ of d, then $a = f_A(x)$, etc. and $a \cap b \cap c \neq \emptyset$, so there is an i such that $i \in a \in \pi_A$, $i \in b \in \pi_B$, $i \in c \in \pi_C$, and therefore $t_i[ABC] = xyz$. Furthermore, by Theorem 3a, Section 4.1, $w \models E_F$.

($\Leftarrow$) Let w be a weak instance for d, $w \models E_F$. Define $\mathcal{I}$ to be $I(w)$ (Definition 5, Sect. 4.1); note that $\mathcal{I}$ satisfies EAP. To see that $\mathcal{I} \models d$, let xyz be a tuple in relation $R[ABC]$ of d: there is a tuple t in w with $t[ABC] = xyz$, and thus $i_t \in f_A(x) \cap f_B(y) \cap f_C(z) = a \cap b \cap c$, i.e., $a \cap b \cap c \neq \emptyset$. Furthermore, by Theorem 3b, Section 4.1, $\mathcal{I} \models E$.

b. Same as (a): all we need to observe is that, if $\mathcal{I}$ satisfies CAD and EAP, then $\mathcal{R}(\mathcal{I})$ has $w[A] = d[A]$ for each A in $\mathcal{U}$. Conversely, if $w[A] = d[A]$ for each A in $\mathcal{U}$, then $\mathcal{I} = I(w)$ satisfies CAD (it always satisfies EAP). ∎

Thus, given d, E we can test in polynomial time whether there is an $\mathcal{I}$ such that $\mathcal{I} \models d, E$, since we can test whether d has a weak instance satisfying E_F [19].

We remark that, since the partition interpretation $\mathcal{I}$ defined from the weak instance w satisfies EAP, introducing EAP as a requirement does not change anything; i.e., there is an $\mathcal{I}$ is an $\mathcal{I}$ such that $\mathcal{I} \models d, E$, EAP iff there is a weak instance for d satisfying E_F. Observe also that, if d consists of a single relation, the two conditions of Theorem 6 collapse into one: $d \models E_F$.

Introducing CAD, even if we have EAP (and thus less possibilities), complicates things. As we will show in Section 6.1, testing for the condition of Theorem 4b is *NP*-complete.

We may now address a more general question. Given a database d and a set of PDs E, we want to know if there is a partition interpretation $\mathscr{I}$ such that $\mathscr{I} \models d, E$.

THEOREM 7. *Let d be a database and E a set of PDs. There is an $\mathscr{I}$ such that $\mathscr{I} \models d, E$ iff there is a weak instance w for d satisfying E.*

Proof. Let w be a weak instance for d satisfying E. It is then easy to see (as in the proof of Theorem 6a) that $I(w) \models d$, and of course by Definition 7, Section 4.1, $I(w) \models E$.

Conversely, let $\mathscr{I}$ be a partition interpretation satisfying d, E. Consider the relation $\mathscr{R}(\mathscr{I})$. As in the Proof of Theorem 6, $\mathscr{R}(\mathscr{I})$ is a weak instance for d, and thus it remains to show that it satisfies E, i.e., that $I(\mathscr{R}(\mathscr{I})) \models E$. Now this partition interpretation $I(\mathscr{R}(\mathscr{I}))$ may be different from $\mathscr{I}$, since it satisfies EAP while $\mathscr{I}$ in general does not. If $\mathscr{I} = \{(\rho_A, \pi_A, f_A) \mid A \in \mathscr{U}\}$, then define $I = \{(\rho, \pi'_A, f'_A) \mid A \in \mathscr{U}\}$, where $\rho = \bigcup_{A \in \mathscr{U}} \rho_A$ and $\pi'_A = \pi_A \cup \{\{x\} \mid x \in \rho - \rho_A\}$. Now the correspondence $\pi'_A \leftrightarrow \pi_A$, $A \in \mathscr{U}$, establishes a *homomorphism* from $L(\mathscr{I})$ to $L(I)$. This is because $\pi'_A \cdot \pi'_B = (\pi_A \cdot \pi_B) \cup \{\{x\} \mid x \in \rho - (\rho_A \cap \rho_B)\}$, and $\pi'_A + \pi'_B = (\pi_A + \pi_B) \cup \{\{x\} \mid x \in \rho - (\rho_A \cup \rho_B)\}$. Since $L(I)$ is the image of $L(\mathscr{I})$ it satisfies all the PDs that $L(\mathscr{I})$ does. Now note that $L(I(\mathscr{R}(\mathscr{I})))$ and $L(I)$ are isomorphic. Therefore, $I(\mathscr{R}(\mathscr{I})) \models E$ since $\mathscr{I} \models E$. ∎

At this point in our exposition we have defined partition semantics and have used relations to measure the expressive power of PDs vs other database dependencies. The connection with weak instance satisfaction, provided by Theorems 6 and 7, indicates that partition semantics handle many relation schemes in an elegant fashion. We will devote the rest of our exposition to issues of complexity. We first examine the complexity of PD implication and then the complexity of testing consistency.

5. IMPLICATION

5.1. *PD Implication and the Uniform World Problem for Lattices*

Given a finite set E of PDs and a PD δ, over attributes $\mathscr{U}$, we want to know if $E \models_{\text{rel}} \delta$, i.e., if δ holds in every relation that satisfies E. We also want to know if $E \models_{\text{rel,fin}} \delta$, i.e., if δ holds in every *finite* relation that satisfies E (recall implication notation from Sects. 2.1, 2.2).

We first observe that these questions can be approached as implication problems for a certain class of algebraic structures, namely lattices with constants over $\mathscr{U}$.

LEMMA 8.1. a. $E \models_{\text{rel}} \delta$ *iff* $E \models_{\text{lat}} \delta$

b. $E \models_{\text{rel,fin}} \delta$ *iff* $E \models_{\text{lat,fin}} \delta$.

Proof. a. ($\Leftarrow$) Suppose $E \models_{\text{lat}} \delta$, and let r be a relation that satisfies E. Then $I(r) \models E$ (Definition 7, Sect. 4.1), and thus the lattice $L(I(r))$ obtained by $I(r)$ by

closing under sums and products satisfies E (Theorem 1, Sect. 3.2). But then δ holds in $I(r)$, and thus r satisfies δ.

($\Rightarrow$) Suppose $E \models_{\text{rel}} \delta$, and let L be a lattice satisfying E. We will show that $L \models \delta$. L is isomorphic to a sublattice of the lattice of partitions of some set ρ [10, 34]. Translated into our terminology, this means that we can find a partition interpretation $\mathscr{I}$, *satisfying* EAP, such that L is isomorphic to $L(\mathscr{I})$. Thus, $\mathscr{I} \models E$. Now consider the relation $\mathscr{R}(\mathscr{I})$: since $\mathscr{I} \models$ EAP, we have that $L(I(\mathscr{R}(\mathscr{I}))) = L(\mathscr{I})$, therefore $\mathscr{R}(\mathscr{I}) \models E$ (Definition 7, Sect. 4.1, and Theorem 1). By the hypothesis that $E \models_{\text{rel}} \delta$ we have therefore that $\mathscr{R}(\mathscr{I}) \models \delta$, which means $I(\mathscr{R}(\mathscr{I})) \models \delta$. Thus $L(\mathscr{I}) \models \delta$, and δ holds in L.

b. ($\Leftarrow$) Observe, in the proof of the "if" direction of (a), that if r is *finite* then $I(r)$ and $L(I(r))$ are also *finite.*

($\Rightarrow$) Observe, in the proof of the "only if" direction of (a), that if L is *finite* then L is isomorphic to a sublattice of the lattice of partitions of some *finite* set ρ [26]. Thus we can find a *finite* partition interpretation $\mathscr{I}$, *satisfying* EAP, such that L is isomorphic to $L(\mathscr{I})$. Since $\mathscr{I}$ is finite, $\mathscr{R}(\mathscr{I})$ is also finite. ∎

Lemma 8.1b is the critical lemma for relating finite implication over relations to finite implication over lattices. The theorem from [26] that is used in the proof, namely: "each finite lattice is isomorphic to a sublattice of the lattice of partitions of some finite set" is nontrivial, and was an important open conjecture for some time in lattice theory [10].

$E \models_{\text{lat}} \delta$ is the uniform word problem for lattices. In Theorem 8 we will argue from first principles that $\models_{\text{lat,fin}}$ is *equivalent* to $\models_{\text{lat}}$. So $\models_{\text{lat,fin}}$ is also the uniform word problem for lattices. This equality of finite and unrestricted implication actually follows from a more general theorem in [25], see also [14, 13].

We now present a complete and sound inference system [12] for the (finite) implication problem for PDs. Suppose we are given a set E of PDs, and a PD $e = e'$: we want to test if $E \models_{\text{lat}} e = e' (E \models_{\text{lat,fin}} e = e')$.

Consider the set $W(\mathscr{U})$ of partition expressions over $\mathscr{U}$, $+, \cdot$. We define five binary relations on $W(\mathscr{U})$: $\leqslant_{\text{id}}$, $=_{\text{id}}$, $\rightarrow\!\rightarrow_E$, $\leqslant_E$, and $=_E$.

I. Define $\leqslant_{\text{id}}$ (*identically* less-than-or-equal) inductively as follows. Here p, q, s are partition expressions over $\mathscr{U}$, and $+, \cdot$ are meant as uninterpreted operations on partition expressions, which return another expression. *ID rules*:

1. $A \leqslant_{\text{id}} A$, A in $\mathscr{U}$;
2. **if** $p \leqslant_{\text{id}} s$ **and** $q \leqslant_{\text{id}} s$ **then** $p + q \leqslant_{\text{id}} s$;
3. **if** $p \leqslant_{\text{id}} s$ **or** $q \leqslant_{\text{id}} s$ **then** $p \cdot q \leqslant_{\text{id}} s$;
4. **if** $s \leqslant_{\text{id}} p$ **and** $s \leqslant_{\text{id}} q$ **then** $s \leqslant_{\text{id}} p \cdot q$;
5. **if** $s \leqslant_{\text{id}} p$ **or** $s \leqslant_{\text{id}} q$ **then** $s \leqslant_{\text{id}} p + q$.

The relation $\leqslant_{\mathrm{id}}$ is reflexive and transitive [10, 33]. Also if $p_1 \leqslant_{\mathrm{id}} q_1$, $p_2 \leqslant_{\mathrm{id}} q_2$, then $p_1 + p_2 \leqslant_{\mathrm{id}} q_1 + q_2$, $p_1 \cdot p_2 \leqslant_{\mathrm{id}} q_1 \cdot q_2$ [10, 33].

II. Define $=_{\mathrm{id}}$ as: $p =_{\mathrm{id}} q$ iff ($p \leqslant_{\mathrm{id}} q$ and $q \leqslant_{\mathrm{id}} p$). The relation $=_{\mathrm{id}}$ is an *equivalence relation*, and in particular it is a *congruence*: i.e., if $p_1 =_{\mathrm{id}} q_1$, $p_2 =_{\mathrm{id}} q_2$, then $p_1 + p_2 =_{\mathrm{id}} q_1 + q_2$ and $p_1 \cdot p_2 =_{\mathrm{id}} q_1 \cdot q_2$. So let $[p]_{\mathrm{id}}$ denote the equivalence class of p in $=_{\mathrm{id}}$. L_{id} is the set of equivalence classes of $=_{\mathrm{id}}$ with two operations $+, \cdot$: $[p]_{\mathrm{id}} + [q]_{\mathrm{id}} = [p+q]_{\mathrm{id}}$, $[p]_{\mathrm{id}} \cdot [q]_{\mathrm{id}} = [p \cdot q]_{\mathrm{id}}$. These operations are well defined because $=_{\mathrm{id}}$ is a congruence.

LEMMA 8.2 [10, 33]. a. *The PD $p = q$ holds in all lattices with constants over $\mathscr{U}$ iff $p =_{\mathrm{id}} q$.*

b. *L_{id} is a lattice.*

This lemma is one of the basic facts from lattice theory. In our case it provides a complete and sound inference system for PD identities.

A simple observation transforms the definition of $\leqslant_{\mathrm{id}}$, above, into a polynomial-time test [6]. Namely, from the structure of the five rules defining $\leqslant_{\mathrm{id}}$, a proof of $p \leqslant_{\mathrm{id}} q$ need only involve subexpressions of p and q. There is only a linear number of such subexpressions. Now apply the five ID rules repeatedly between all triples of these subexpressions, until no rule is applicable; this is a polynomial-time process.

III. We now wish to capture the effect of the equalities E. Define relation $\rightarrow\rightarrow_E$ on $W(\mathscr{U})$ as follows: $p \rightarrow\rightarrow_E q$ iff (there is some $n \geqslant 0$ and a sequence $s_0, \ldots, s_n$ of expressions such that $s_0 = p$, $s_n = q$, and for $0 \leqslant i \leqslant n-1$, s_{i+1} is obtained by an E-substitution on s_i). A E-substitution on expression s_i is a replacement of an occurence of a subexpression z of s_i by expression v, where $z = v$ or $v = z$ is in E.

It is clear that $\rightarrow\rightarrow_E$ is reflexive, symmetric, and transitive. Thus it is an equivalence relation. One can also easily see that if $p_1 \rightarrow\rightarrow_E q_1$, $p_2 \rightarrow\rightarrow_E q_2$, then $p_1 + p_2 \rightarrow\rightarrow_E q_1 + q_2$, $p_1 \cdot p_2 \rightarrow\rightarrow_E q_1 \cdot q_2$. Therefore, $\rightarrow\rightarrow_E$ is a congruence.

The $\rightarrow\rightarrow_E$ congruence is the one used in [23] in order to provide a polynomial-time solution to the *uniform word problem for finitely presented algebras.*

IV. Our fourth step is to combine $\leqslant_{\mathrm{id}}$ and $\rightarrow\rightarrow_E$. Define $\leqslant_E$ as the *sum* of $\leqslant_{\mathrm{id}}$, $\rightarrow\rightarrow_E$: $p \leqslant_E q$ iff (there is some $n \geqslant 0$ a sequence of expressions $s_0, \ldots, s_n$ such that $p = s_0$, $s_n = q$, and for $i = 0, \ldots, n-1$, $s_i \leqslant_{\mathrm{id}} s_{i+1}$ or $s_i \rightarrow\rightarrow_E s_{i+1}$).

It is easy to see that $\leqslant_E$ is reflexive and transitive. Also if $p_1 \leqslant_E q_1$, $p_2 \leqslant_E q_2$, then $p_1 + p_2 \leqslant_E q_1 + q_2$, $p_1 \cdot p_2 \leqslant_E q_1 \cdot q_2$, because both $\leqslant_{\mathrm{id}}$ and $\rightarrow\rightarrow_E$ have this property; this argument is quite common in universal algebra [18].

V. Define $=_E$ as follows: $p =_E q$ iff ($p \leqslant_E q$ *and* $q \leqslant_E p$).

From its definition and the properties of $\leqslant_E$, it follows that: the relation $=_E$ is an equivalence relation and a congruence. Since $=_E$ is a congruence, the operations $\cdot$, $+$ are well defined on the equivalence classes of $=_E$. If p is a partition expression

let $[p]_E$ denote its equivalence class in $=_E$. Let L_E be the set of equivalence classes of $=_E$, with the operations $[p]_E + [q]_E = [p+q]_E$ and $[p]_E \cdot [q]_E = [p \cdot q]_E$.

LEMMA 8.3. *L_E is a lattice.*

Proof. Just check the LA from Section 2.2; e.g., $[p]_E + [p]_E = [p+p]_E = [p]_E$, because $p + p =_{\mathrm{id}} p$ (Lemma 8.2b), and therefore $p + p =_E p$. In general if $p =_{\mathrm{id}} q$ then $p =_E q$, and this makes the check of LA straightforward, given that L_{id} is a lattice [10, 33]. ∎

We now show that the relation $=_E$ captures the PDs (finitely) implied by E. The equivalence of Theorem 8b and c follows from [25]:

THEOREM 8. *The following statements are equivalent*:

a. $e =_E e'$

b. $E \models_{\mathrm{lat}} e = e'$

c. $E \models_{\mathrm{lat,fin}} e = e'$

d. $E \models_{\mathrm{rel}} e = e'$

e. $E \models_{\mathrm{rel,fin}} e = e'$.

Proof. Observe that, from the way $\leqslant_{\mathrm{id}}$ and $\leqslant_E$ were defined, if $e \leqslant_E e'$ then $e \leqslant e'$ in every lattice satisfying E (where $\leqslant$ is the partial order of the lattice). Thus, (a) $\Rightarrow$ (b).

To prove (b) $\Rightarrow$ (a), we claim that L_E satisfies a PD $p = q$ iff $p =_E q$. From Lemmas 8.2 and 8.3 we have that L_E is a lattice over constants in $\mathcal{U}$, because constant or attribute A is the name of $[A]_E$, etc. If $p =_E q$ then $[p]_E = [q]_E$; in other words, the partition expressions p, q are names for the same element of L_E and thus $L_E \models p = q$. If $p \neq_E q$ then, because $[\cdot]_E$ are equivalence classes, $[p]_E \neq [q]_E$; in other words, partition expressions p, q are names for different elements of L_E and thus L_E does not satisfy $p = q$. If $e \neq_E e'$ then L_E does not satisfy $e = e'$, whereas it satisfies E; i.e., L_E is a *counterexample* to $E \models_{\mathrm{lat}} e = e'$. This completes (b) $\Rightarrow$ (a).

We now show the equivalence of (b), (c). The direction (b) $\Rightarrow$ (c) is obvious. To prove the converse, we adapt an argument of [11] (see also [10]), originally given for the special case $E = \varnothing$. Suppose E does not imply $e = e'$; we will show that there is a *finite* lattice which satisfies E but violates $e = e'$. Let $\{A_i \mid i = 1, \ldots, n\}$ be the set of attributes appearing in E, e, e'. Let H be the set of all partition expressions (over the A_i's) of complexity at most as high as the maximum complexity of e, e' and the expressions in E, where by complexity we mean the number of instances of $+, \cdot$. Note that H is finite, since E is finite. Consider now the subset L_H of L_E consisting of all finite products of the equivalence classes (under $=_E$) of elements of H, together with the equivalence class of $A_1 + \cdots + A_n$. It is not hard to verify that L_H is a *sublattice* of L_E. The verification of this is identical with that of [11].

But by the equivalence of (a), (b), $e \neq_E e'$, so L_H satisfies E and violates $e = e'$.

Note also that L_H is finite, because the number of its elements is at most as large as the number of subsets of H (by the properties of $\cdot$). This completes (c) $\Rightarrow$ (b).

The equivalence of (d), (e) with (a), (b), (c) follows from Lemma 8.1. ∎

Our definition of $=_E$, based on $\leqslant_{\mathrm{id}}$ and $\rightarrow\rightarrow_E$, has provided us with a complete and sound inference system for the uniform word problem for lattices (Theorem 8). Note that we also have that: $\models_{\mathrm{lat}}$ is equivalent to $\models_{\mathrm{lat,fin}}$. This equivalence of finite and unrestricted implication demonstrates that the uniform word problem for lattices is decidable. However, the decision algorithm in [25, 14, 13] runs in exponential time. We now modify our inference system to produce a polynomial time decision procedure.

5.2. *An Efficient Algorithm for PD Implication*

In order to test PD implication in polynomial time it clearly is sufficient to test in polynomial time, given E, e, e', whether $e \leqslant_E e'$. Let V be the set of all subexpressions of e, e' and the expressions appearing in E. We construct a set Γ of directed arcs, over V, using the following algorithm:

ALGORITHM ALG.

begin
$\Gamma \leftarrow \varnothing$
repeat until no new arcs are added
1. Add (A, A), $A \in \mathscr{U}$
2. **if** $(p, s) \in \Gamma$ **and** $(q, s) \in \Gamma$ **and** $p + q \in V$
then add $(p + q, s)$ to Γ
3. **if** $((p, s) \in \Gamma$ **or** $(q, s) \in \Gamma)$ **and** $p \cdot q \in V$
then add $(p \cdot q, s)$ to Γ
4. **if** $(s, p) \in \Gamma$ **and** $(s, q) \in \Gamma$ **and** $p \cdot q \in V$
then add $(s, p \cdot q)$ to Γ
5. **if** $((s, p) \in \Gamma$ **or** $(s, q) \in \Gamma)$ **and** $p + q \in V$
then add $(s, p + q)$ to Γ
6. **if** $p = q$ in E **then** add (p, q) **and** (q, p) to Γ
7. **if** $(p, s) \in \Gamma$ **and** $(s, q) \in \Gamma$
then add (p, q) to Γ

end
end

Observe that steps 1–5 in the Algorithm ALG mirror the definition of $\leqslant_{\mathrm{id}}$, i.e., the ID rules restricted to subexpressions of E, e, and e'.

Claim. For Γ defined by ALG and $p, q \in V$, $p \leqslant_E q$ iff $(p, q) \in \Gamma$.

Clearly, proving this claim will demonstrate that testing PD (finite) implication can be done in polynomial time. Because, to test if $e \leqslant_E e'$, construct the digraph (V, Γ) and see if it has an arc from e to e'. This can be done in $O(n^4)$ time. To prove the claim we study a set of *rewrite rules* [21] for $\leqslant_E$.

Rewrite Rules (RR). Take the union, over all pairs of partiton expressions x, y over $\mathcal{U}$, of the following *rules*:

1. $x + x \to\to x$
2. $x \cdot y \to\to x$
3. $y \cdot x \to\to x$
4. $x \to\to x \cdot x$
5. $x \to\to x + y$
6. $x \to\to y + x$
7. $z \to\to v$, where $z = v$ or $v = z$ is in E.

We say that $p \to\to q$, when q is obtained from p by replacing an occurence of a subexpression s of p, which is the right-hand of a rule $s \to\to s'$ in RR, by the expression s', which is the left-hand side of this rule. We say that partition expression p can be *rewritten* by RR as partition expression q (notation: $p \to\to_{RR} q$), when there is some $n \geqslant 0$ and a sequence of expressions $s_0,..., s_n$ such that $p = s_0$, $s_n = q$, and for $i = 0,..., n-1$ we have $s_i \to\to s_{i+1}$.

Note that $\to\to_{RR}$ is the reflexive, transitive closure of $\to\to$ on $W(\mathcal{U})$. Also $\to\to_E$ (defined in the previous section) is the reflexive, transitive closure of $\to\to$ restricted to rules of type 7. Also note that in RR of types 5 and 6 above the right-hand side have subexpression y, which does not appear in the left-hand side.

LEMMA 9.1. *If* $p \leqslant_E q$, *then* $p \to\to_{RR} q$.

Proof. An easy induction shows that, if $p \leqslant_{id} q$, then p can be rewritten by RR as q, using rules 1–6. To see this consider

ID rule 1: trivial.

ID rule 2: if p and q can be rewritten by RR as s, then $p + q$ can be rewritten by RR as $s + s$, and $s + s$ can be rewritten by an RR of type 1 as s.

ID rule 3: if p or q can be rewritten by RR as s, then by RR of types 2, 3, $p \cdot q$ can be rewritten as p or q, and then rewritten as s.

ID rule 4: if s can be rewritten by RR as p and q, then by an RR of type 4, s can be rewritten as $s \cdot s$, and then rewritten as $p \cdot q$.

ID rule 5: if s can be rewritten by RR as p or q, then by also using RR of types 5, 6, s can be rewritten to $p + q$.

Note that $\to\to_E$ is really produced by rewriting with RR of type 7. It is now clear that by the definition of $\leqslant_E$, if $p \leqslant_E q$ then there is a sequence of expressions $s_0,..., s_n$ such that $p = s_0$, $s_n = q$, and for $i = 0,..., n-1$ we have $s_i \to\to s_{i+1}$. ∎

We can now prove our previous *claim*. This is a central lemma in our exposition.

LEMMA 9.2. *Let* Γ *be defined by ALG and* $p, q \in V$. *Then* $p \leqslant_E q$ *iff* $(p, q) \in \Gamma$.

Proof. It is straightforward to see that every arc added by ALG corresponds to a sound inference. Thus if $(p, q) \in \Gamma$ then $p \leqslant_E q$. If $p \leqslant_E q$ then, by Lemma 9.1, there is a sequence of expressions $s_0, \ldots, s_n$ such that $p = s_0$, $s_n = q$, and for $i = 0, \ldots, n-1$ we have $s_i \rightarrow\!\!\rightarrow s_{i+1}$. We call such a sequence a *proof* that $p \leqslant_E q$. We wish to show: for $p, q \in V$ if $p \leqslant_E q$ then $(p, q) \in \Gamma$. By Lemma 9.1 it suffices to show that for $p, q \in V$, if there is a proof of $p \leqslant_E q$ then $(p, q) \in \Gamma$.

Now we define a relation $\prec$ on pairs of expressions: $(p_1, q_1) \prec (p_2, q_2)$ iff $p_1 \leqslant_E q_1$, $p_2 \leqslant_E q_2$, and either

(i) the shortest length proof that $p_1 \leqslant_E q_1$ is shorter than the shortest length proof that $p_2 \leqslant_E q_2$, or

(ii) the shortest proofs that $p_1 \leqslant_E q_1$, $p_2 \leqslant_E q_2$ have the same length, and p_1 is a proper subexpression of p_2, and q_1 is a proper subexpression of q_2.

Clearly $\prec$ is a well-founded partial order (no infinite descending chains). We proceed by induction on $\prec$:

Basis. There is a proof that $p \leqslant_E q$ of length 0. Then $p = q$, and $(p, q) \in \Gamma$.

Induction step. Let $p, q \in V$, and assume that the *claim* holds for $p', q' \in V$ whenever $(p', q') \prec (p, q)$. We will show that the claim holds for (p, q). Let $s_0, \ldots, s_n$, $n > 0$, be a shortest length proof that $p \leqslant_E q$.

Case 1. For each $i = 0, \ldots, n-1$. s_{i+1} is obtained from s_i by replacing a *proper* subexpression of s_i according to RR 1–7. Then $p = p_1 \theta p_2$, $q = q_1 \theta q_2$ $(\theta \in \{+, \cdot\})$, where $p_i \leqslant_E q_i$ via proofs at most as long as the proof that $p \leqslant_E q$, and p_i (q_i) is a proper subexpression of p (q). Thus $(p_i, q_i) \prec (p, q)$, and furthermore $p_i, q_i \in V$, so by the induction hypothesis $(p_i, q_i) \in \Gamma$. It then easily follows that $(p, q) \in \Gamma$.

Case 2. For some i, $0 \leqslant i \leqslant n-1$, s_i is rewritten into s_{i+1} according to one of the RR $1-7$ (i.e., $s_i \rightarrow\!\!\rightarrow s_{i+1}$ and the left-hand side of the RR used is s_i and not one of its proper subexpressions). Let us call this set of i's the index set I. There are two possibilities:

Case 2a. For some i in I the RR used is of type 7. This means p is rewritten as z, $z = v$ $(v = z)$ is in E, and v is rewritten as q. Then clearly $(p, z) \prec (p, q)$ and since $z \in V$, by the induction hypothesis $(p, z) \in \Gamma$. Similarly $(v, q) \in \Gamma$. It follows that $(p, q) \in \Gamma$.

Case 2b. For all i in I, the RR used is of the type 1–6. We consider the *smallest* such i (let us call it min) and the *largest* such i (let us call it max). We distinguish six subcases according to which type of rule was used to rewrite s_i as s_{i+1}, for these special i's in I. We then show that these six subcases exhaust Case 2b:

Rule used for min is of type 1. This means that there exists an s such that $p = p_1 + p_2$, p_1 is rewritten as s, p_2 is rewritten as s, and s is rewritten as q. Then $p_i \leqslant_E q$ via proofs shorter than the proof that $p \leqslant_E q$, so $(p_i, q) \prec (p, q)$. Also $p_i \in V$, so by the induction hypothesis $(p_i, q) \in \Gamma$. It follows that $(p, q) \in \Gamma$.

Rule used for min is of type 2. This means that there exists an s such that $p = p_1 \cdot p_2$, p_1 is rewritten as s, s is rewritten as q. Then $p_1 \leqslant_E q$ via a proof shorter than the proof that $p \leqslant_E q$, so $(p_1, q) \prec (p, q)$. Also $p_1 \in V$, so by the induction hypothesis $(p_1, q) \in \Gamma$. It follows that $(p, q) \in \Gamma$.

Rule used for min is of type 3. Similar to previous subcase.

Rule used for max is of type 4. This means that there exists an s such that $q = q_1 \cdot q_2$, s is rewritten as q_1, s is rewritten as q_2, and p is rewritten as s. Then $p \leqslant_E q_i$ via shorter than the proof that $p \leqslant_E q$, so $(p, q_i) \prec (p, q)$. Also $q_i \in V$, so by the induction hypothesis $(p, q_i) \in \Gamma$. It follows that $(p, q) \in \Gamma$.

Rule used for max is of type 5. This means that there exists an s such that $q = q_1 + q_2$, s is rewritten as q_1, p is rewritten as s. Then $p \leqslant_E q_1$ via a proof shorter than the proof that $p \leqslant_E q$, so $(p, q_1) \prec (p, q)$. Also $q_1 \in V$, so by the induction hypothesis $(p, q_1) \in \Gamma$. It follows that $(p, q) \in \Gamma$.

Rule used for max is of type 6. Similar to previous subcase.

At this point all we have to do to complete the induction is to argue that: in a shortest proof of $p \leqslant_E q$ of the Case 2b form, it is impossible to have type 4, 5, or 6 used for min and type 1, 2, or 3 used for max. For suppose that this were so, then there would be j, k in I (two consecutive indices in I), such that

p is rewritten as s_j,

s_j is rewritten as s_{j+1} using a rule (rl_j) of type 4, 5, or 6,

s_{j+1} is rewritten as s_k as in Case 1,

s_k is rewritten as s_{k+1} using a rule (rl_k) of type 1, 2, or 3,

s_{k+1} is rewritten as q.

Because of the operator symbol at the top, the only possibilities are: (1) rl_j is of type 4 and rl_k of type 2 or 3, (2) rl_j is of type 5 or 6 and rl_k of type 1. For each one of these possibilities one can easily shorten the (assumed to be shortest) proof of $p \leqslant_E q$ by one step. This is the desired contradiction, which completes Case 2b and the proof of this lemma. ∎

The proof of this lemma, which privdes an efficient algorithm for the uniform word problem for lattices, is an extension of the proof of Theorem 1 in [23], which provides an efficient algorithm for the uniform word problem for finitely presented algebras. Lemmas 9.1 and 9.2 demonstrate that ALG can be used as an $O(n^4)$ algorithm for PD implication. Therefore we have shown

THEOREM 9. *There is a polynomial time algorithm for the (finite) implication problem for PDs.*

5.3. *Special PDs: FPDs and PD Identities*

Let δ_σ be the FPD corresponding to an FD σ (i.e., δ_σ is $X = X \cdot Y$ if σ is $X \to Y$). Also, let E_Σ be the set of FPDs corresponding to a set of FDs Σ. Since $r \models \sigma$ iff

$r \models \delta_\sigma$ we have $\Sigma \models_{\text{rel}} \sigma$ iff $E_\Sigma \models_{\text{rel}} \delta_\sigma$. Therefore, the implication problem for FDs can be reduced, in a straightforward way, to the *uniform word problem for idempotent commutative semigroups* (structures with a single operator $\cdot$, which is associative, commutative, and idempotent). On the other hand, since $X = Y$ is equivalent to $X = X \cdot Y$ **and** $Y = Y \cdot X$ (see Example f, Sect. 4.2), the uniform word problem for idempotent commutative semigroups can be reduced directly of FD implication.

The complete inference system of [2] and the efficient algorithms in [3] (for FD implication) are therefore, directly applicable to the uniform word problem for idempotent commutative semigroups.

Since inference of FDs can be seen as a special case of inference of PDs, the problem of PD implication is actually *polynomial-time complete* [31]. However, in the special case where E is empty [10, 33] it can be solved in *logarithmic space* [20], as we now outline.

THEOREM 10. *The problem of recognizing PD identities is solvable in logarithmic space.*

Proof. Clearly it suffices to describe how to recognize $\leqslant_{\text{id}}$ in logarithmic space. First, observe:

1. $A \leqslant_{\text{id}} A'$ **iff** (A is identical to A', A, A' in $\mathcal{U}$).
2. $A \leqslant_{\text{id}} p' \cdot q'$ **iff** ($A \leqslant_{\text{id}} p'$ **and** $A \leqslant_{\text{id}} q'$, A in $\mathcal{U}$).
3. $A \leqslant_{\text{id}} p' + q'$ **iff** ($A \leqslant_{\text{id}} p'$ **or** $A \leqslant_{\text{id}} q'$, A in $\mathcal{U}$).
4. $p \cdot q \leqslant_{\text{id}} A'$ **iff** ($p \leqslant_{\text{id}} A'$ **or** $q \leqslant_{\text{id}} A'$, A' in $\mathcal{U}$).
5. $p \cdot q \leqslant_{\text{id}} p' \cdot q'$ **iff** ($p \cdot q \leqslant_{\text{id}} p'$ **and** $p \cdot q \leqslant_{\text{id}} q'$).
6. $p \cdot q \leqslant_{\text{id}} p' + q'$ **iff** ($p \leqslant_{\text{id}} p' + q'$ **or** $q \leqslant_{\text{id}} p' + q'$ **or** $p \cdot q \leqslant_{\text{id}} p'$ **or** $p \cdot q \leqslant_{\text{id}} q'$).
7. $p + q \leqslant_{\text{id}} e'$ **iff** ($p \leqslant_{\text{id}} e'$ **and** $q \leqslant_{\text{id}} e'$).

In each of the above cases, the "if" direction is trivial. The "only if" direction follows in case 5 because $p' \cdot q' \leqslant_{\text{id}} p'$ and $p' \cdot q' \leqslant_{\text{id}} q'$, and in case 7 because $p \leqslant_{\text{id}} p + q$, $q \leqslant_{\text{id}} p + q$. In the remaining cases, the "only if" direction follows by the definition of $\leqslant_{\text{id}}$.

The above observation gives a *recursive* algorithm to test, given e, e', whether $e \leqslant_{\text{id}} e'$. We now describe how to implement this recursion using only logarithmic auxiliary space.

First, note that the results of intermediate recursive calls need not be stored. For example, consider case 7: if the recursive call for $p \leqslant_{\text{id}} e'$ returns *false*, then we immediately return *false*; otherwise, we return the result of the recursive call for $q \leqslant_{\text{id}} e'$.

We will also argue that we do not need to store the arguments of previous recursive calls. Thus, all we need to have in storage at any particular point is the

arguments of the recursive call which is being evaluated. Since the arguments are *subexpressions* of e, e', we can just have two *pointers* to the appropriate places in the input, and this only takes logarithmic space.

We will now describe how, given two pointers to two subexpressions p, p' of e, e', respectively, we can find the next recursive call to be evaluated using only logarithmic additional space. We assume that e, e' are represented (in the standard way) as binary trees, so that, given a pointer to a node u, we can find a pointer to the father (right son, left son) of u.

We use two auxiliary pointers α, α', initialized to the root of e, e', respectively. Let $\mathscr{C}(e, e')$ be the set of recursive calls generated from the call $e \leqslant_{\mathrm{id}} e'$ ($\mathscr{C}(e, e')$ contains either two or four members, depending on which of cases 2–7 is the relevant one). We will show that we can determine which member of $\mathscr{C}(e, e')$ eventually gives rise to the call $p \leqslant_{\mathrm{id}} p'$, using only logarithmic additional space. If this member of $\mathscr{C}(e, e')$ turns out to be the call $e_1 \leqslant_{\mathrm{id}} e_1'$, we set the pointers α, α' to the expressions e_1, e_1' respectively and we repeat with $\mathscr{C}(e_1, e_1')$. Continuing in this way, we will eventually find e_i, e_i' such that the call $p \leqslant_{\mathrm{id}} p'$ is in $\mathscr{C}(e_i, e_i')$. We can then easily determine the next call to be evaluated.

Finally, note that, to determine which member of $\mathscr{C}(e, e')$ eventually gives rise to the call $p \leqslant_{\mathrm{id}} p'$, we only need to know whether p (p') is in the left or in the right subtree of e (e'). This can be found by walking the tree representing e in a depth-first fashion, until we encounter p. This walk can be done using only logarithmic additional space, because all we need to remember is the node v which is currently visited and the node z which was visited immediately before v: if z is the father of v, we next visit the left son of v; if z is the left son of v, we next visit the right son of v; if z is the right son of v, we next visit the father of v. ∎

6. Testing Consistency

6.1. *The Complete Atomic Data Assumption*

Let d be a database over attributes $\mathscr{U}$, and E a set of FPDs. We will demonstrate that any consistency test in the presence of CAD and EAP is unlikely to be efficient. Because of Theorem 6b, Section 4.3, this test is equivalent to testing whether there exists a weak instance w satisfying E and with $w[A] = d[A]$ for each A in $\mathscr{U}$.

Theorem 11. *Given a database d and a set of FPDs E, it is NP-complete to test whether there is an $\mathscr{I}$ such that $\mathscr{I} \models d$, E and $\mathscr{I} \models$ CAD, EAP.*

Proof. Membership in *NP* follows from Theorem 6b; just guess an appropriate weak instance w. Since E consists only of FPDs (i.e., of FDs) w need only contain one tuple for each tuple of d. *NP*-hardness is shown by a reduction from NOT-ALL-EQUAL-3SAT [17]; given a 3CNF Boolean formula φ over variables $x_1, \ldots, x_n$ with clauses $c_1, \ldots, c_m$, test whether there is a truth assignment under which each clause c_i has one true and one false literal.

From φ construct d, E as follows: d has a relation $R_0[AA_1 \cdots A_n]$ with two tuples $au_1 \cdots u_n$, $av_1 \cdots v_n$, and for each clause of φ, say $c_1 = x_1 \vee x_2 \vee (\neg x_3)$, d has a relation $R_1[AA_4 \cdots A_n B_1 \cdots B_n]$, with a single tuple $by_4 \cdots y_n a_1 a_2 b_3 z_4 \cdots z_n$. Note that the scheme R_1 does not have attributes A_1, A_2, A_3, corresponding to the variables of the clause c_1. Also the structure of the single tuple depends on which variables appear negated and which not in c_1 (in this case ... $a_1 a_2 b_3$...).

E contains $B_i = B_i \cdot A_i$, $i = 1,..., n$, and for clause c_1 it contains $B_1 \cdot B_2 \cdot B_3 = B_1 \cdot B_2 \cdot B_3 \cdot A$ (and similar FPDs for other clauses). Thus, E_F (the FDs equivalent to the FPDs E) consists of $B_i \to A_i$, $i = 1,..., n$, and $B_1 B_2 B_3 \to A$ for clause c_1, etc. Figure 3 shows an example for $n = 4$.

We now show that φ is satisfiable (by an assignment leaving one literal of each clause false) iff relation $R[AA_1 \cdots A_n B_1 \cdots B_n]$ (see Fig. 3) can be filled in so that no new symbols are introduced in any column, and the FDs in E_F are satisfied (by Theorem 6b, this is equivalent to existence of an $\mathscr{I}$ such that $\mathscr{I} \models d$, E, CAD, EAP).

Observe that for each i, $t_1[B_i] \neq t_2[B_i]$, because of the FD $B_i \to A_i$. We can also make sure that for each variable x_i there is a clause of φ which does not contain x_i (just add a clause $x_{n+1} \vee (\neg x_{n+1})$, where x_{n+1} is a *new* variable). This does not affect the *NP*-completeness of the original question on φ. However it forces $t_1[B_i] \neq z_i$, $t_2[B_i] \neq z_i$ (again because of the FD $B_i \to A_i$ and because u_i, v_i, y_i are distinct). Thus $\{t_1[B_i], t_2[B_i]\} = \{a_i, b_i\}$. Make variable x_i *true* if $t_1[B_i] = a_i$, *false* if $t_1[B_i] = b_i$. Because of the FD $B_1 B_2 B_3 \to A$ and the tuple t_3, we will only be able to fill in the rest of the values iff the above truth assignment makes one literal

R_0 :

A	A_1	A_2	A_3	A_4
a	u_1	u_2	u_3	u_4
a	v_1	v_2	v_3	v_4

R_1 :

A	A_4	B_1	B_2	B_3	B_4
b	y_4	a_1	a_2	b_3	z_4

R :

	A	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
t_1:	a	u_1	u_2	u_3	u_4				
t_2:	a	v_1	v_2	v_3	v_4				
t_3:	b				y_4	a_1	a_2	b_3	z_4

E_F : $B_i \to A_i$, $i = 1, \ldots, 4$

$B_1\ B_2\ B_3 \to A$

$C_1 = x_1 \vee x_2 \vee (\neg x_3)$

FIGURE 3

of c_1 true and one false. This is because we are free to fill in the A_i columns, but in the B_i columns of t_1 and t_2 we are appropriately constrained. ∎

6.2. *An Efficient Consistency Test*

Our final goal is: given a database d over attributes $\mathcal{U}$ and a set of PDs E, to test whether there exists a satisfying partition interpretation $\mathcal{I}$. By Theorem 7, Section 4.3, this is equivalent to testing whether there exists a weak instance for d satisfying E.

First, we replace E by a set E' of PDs of the form $C = A \cdot B$ or $C = A + B$, where A, B, C are attributes from a universe $\mathcal{U}'$ containing $\mathcal{U}$; this is done by (recursively) replacing $X = Y \cdot Z$ or $X = Y + Z$ by the PDs $X = C$, $Y = A$, $Z = B$, $C = A \cdot B$, or $C = A + B$, where A, B, C are *new* attribute names. It is easy to check that there is a weak instance for d satisfying E iff there is a weak instance for d satisfying E'.

Now let us replace certain PDs in E' with equivalent PDs. A PD $C = A \cdot B$ in E' can be replaced by the FPDs $C \leqslant A \cdot B$, $A \cdot B \leqslant C$. A PD $C = A + B$ in E' can be replaced by the PDs $A + B \leqslant C$ and $C \leqslant A + B$. Furthermore, the PD $A + B \leqslant C$ can be replaced by the FPDs $A \leqslant C$, $B \leqslant C$. We now have transformed E' into an equivalent form consisting of FPDs and of PDs of the form $C \leqslant A + B$. After this transformation we also have that: there is a weak instance for d satisfying E iff there is a weak instance for d satisfying E'.

Now compute (using the algorithm of the previous section) all *consequences* of E' of the form $A \leqslant B$, A, B in $\mathcal{U}'$, and add them to E'. Call this closure E^+. Notice that if E^+ contains $A \leqslant B$ and $C \leqslant A + B$ ($C \leqslant B + A$) it must also contain $C \leqslant B$: in this case delete $C \leqslant A + B$ ($C \leqslant B + A$) from E^+. The E^+ we have thus constructed consists of a set of FPDs and a set of PDs of the form $C \leqslant A + B$, where neither $A \leqslant B$ nor $B \leqslant A$ are in E^+.

It is easy to see that after all these transformations, there is a weak instance for d satisfying E iff there is a weak instance for d satisfying E^+. Let F be the set of FPDs in E^+. The crucial fact is given in the following.

LEMMA 12.1. *There is a weak instance for d satisfying E^+ iff there is a weak instance for d satisfying F.*

Proof. The "only if" direction is obvious. For the converse, let w be a weak instance for d satisfying F. Suppose some PD $C \leqslant A + B$ in E^+ is violated by tuples t_1, t_2 of w, where $t_1[ABC] = a_1 b_1 c$, $t_2[ABC] = a_2 b_2 c$, $a_1 \neq a_2$, $b_1 \neq b_2$. We can remedy this violation by adding to w a *new* tuple such that $t[AB] = a_1 b_2$. To make sure that the relation w_1 obtained still satisfies F, let $A^+ = \{X \mid F \models A \leqslant X\}$, $B^+ = \{X \mid F \models B \leqslant X\}$: we make $t[A^+] = t_1[A^+]$, $t[B^+] = t_2[B^+]$, and fill in the rest of the attributes of t with distinct new values (not appearing in w). To argue that this is indeed possible, observe first that B is no in A^+ and A is not in B^+ (otherwise $C \leqslant A + B$ would not appear in E^+). We also have to make sure that, if $Q \in A^+$ and $Q \in B^+$, then $t_1[Q] = t_2[Q]$. But if Q appears in both A^+ and B^+ we have $F \models A \leqslant Q$, $F \models B \leqslant Q$, so since $C \leqslant A + B$ is in E^+ we have $E^+ \models C \leqslant Q$,

and therefore $C \leqslant Q$ is in F. This implies that $t_1[Q] = t_2[Q]$, since $t_1[C] = t_2[C]$ and w satisfies F.

We now repeat the above argument, starting with w_1, to obtain relations w_2, w_3 and so on. The relation w_ω obtained after an infinite number of steps is a weak instance for d satisfying E^+, because any violation of some PD $C \leqslant A + B$ appearing at any stage has been taken care of at some later stage. ∎

We can now prove

THEOREM 12. *There is a polynomial-time algorithm to test whether a given database d is consistent with a set E of PDs.*

Proof. Using the polynomial-time algorithm for inference of PDs given in Section 5, we can construct the set F. By Lemma 12.1, we can then use the chase algorithm of [19] to test if d is consistent with F. ∎

7. CONCLUSIONS

We have shown that: (1) the inference problem for PDs and (2) the problem of testing consistency of a set of relations with a set of PDs are in polynomial time. Both proofs use algebraic techniques and make use of finite and infinite relations. If we restrict ourselves to finite relations then (1) is still in polynomial time (by the same algorithm), but (2) remains open.

We would like to point out that the FD implication problem can be formulated, in a straightforward fashion, as a special case of the generator problem for finitely presented algebras [23, 9]. In fact, there are many similarities between the efficient algorithm for FD implication of [3] and the algorithm of [23] for this generator problem. Our analysis here reveals much more of this problem's algebraic nature. PD implication is the uniform word problem for lattices, and FD implication the uniform word problem for idempotent commutative semigroups.

Finally, we would like to emphasize that even if we assign partition semantics to the relational data model, we still can use all the familiar algebraic operations on relations (selection, projection, cartesian product, union, difference, etc.). After all these operations are syntactic manipulations of syntactic objects. What we gain, on the other hand, is a more meaningful treatment of databases over many relation schemes.

ACKNOWLEDGMENTS

We would like to thank Richard Hull and Moshe Vardi for their many helpful comments.

REFERENCES

1. A. V. AHO AND J. D. ULLMAN, Universality of data retrieval languages, *in* "Proceedings, 6th ACM Symp. Principles of Programming Languages," 1979, pp. 110–120.
2. W. W. ARMSTRONG, Dependency structure of database relationships, *in* "Proceedings, IFIP–74," Amsterdam, 1974, pp. 580–583.
3. C. BEERI AND P. A. BERNSTEIN, Computational problems related to the design of normal form relational schemas, *ACM Trans. Database Systems* **4**, No. 1 (1979), 30–59.
4. C. BEERI AND M. Y. VARDI, Formal systems for tuple and equality generating dependencies, *SIAM J. Comput.* **13**, No. 1 (1984), 76–98.
5. C. BEERI AND M. Y. VARDI, A proof procedure for data dependencies, *J. Assoc. Comput. Mach.* **31**, No. 4 (1984), 718–741.
6. P. A. BLONIARZ, H. B. HUNT, III, AND D. J. ROSENKRANTZ, Algebraic structures with hard equivalence and minimization problems. *J. Assoc. Comput. Match.* **31**, No. 4 (1984), 879–904.
7. M. A. CASANOVA, R. FAGIN, AND C. H. PAPADIMITRIOU, Inclusion dependencies and their interaction with functional dependencies, *J. Comput. System Sci.* **28**, No. 1 (1984), 29–59.
8. S. S. COSMADAKIS, "Equational Theories and Database Constraints," Ph. D. thesis MIT, 1985.
9. S. S. COSMADAKIS AND P. C. KANELLAKIS, Equational theories and database constraints, *in* "Proceedings, 17th ACM Sympos. Theory of Computing," May, 1984, pp. 273–284.
10. P. CRAWLEY AND R. P. DILWORTH, "Algebraic Theory of Lattices," Prentice–Hall, Englewood Cliffs, N.J., 1973.
11. R. A. DEAN, Component subsets of the free lattice on *n* generators, *Proc. Amer. Math. Soc.* 7 (1956), 220–226).
12. H. B. ENDERTON, "A Mathematical Introduction to Logic," Academic Press, New York/London, 1972.
13. T. EVANS, The word problem for abstract algebras, *J. London Math. Soc.* **26** (1951), 64–71.
14. T. EVANS, Word problems, *Bull. Amer. Math. Soc.* **84** (1978), 789–802.
15. R. FAGIN, Horn clauses and database dependencies, *J. Assoc. Comput. Mach.* **29**, No. 4 (1982), 952–985.
16. R. FAGIN, Generalized first-order spectra and polynomial-time recognizable sets, *SIAM-AMS Proc.* **7**, No. 1 (1974), 43–73.
17. M. R. GAREY AND D. S. JOHNSON, "Computers and Intractability: A Guide to the Theory of *NP*-Completeness," Freeman, San Francisco, 1979.
18. G. GRÄTZER, "Universal Algebra," Springer-Verlag, New York, 1979.
19. P. HONEYMAN, Testing satisfaction of functional dependencies, *J. Assoc. Comput. Mach.* **29**, No. 3 (1982), 668–677.
20. J. E. HOPCROFT AND J. D. ULLMAN, "Introduction to Automata Theory, Languages, and Computation," Addison–Wesley, Reading, Mass., 1979.
21. G. HUET AND D. OPPEN, Equations and rewrite rules: A survey, *in* "Formal Languages: Perspectives and Open Problems" (R. Book, Ed.), pp. 349–403, Academic Press, New York/London, 1980.
22. H. B. HUNT, III, D. J. ROSENKRANTZ, AND P. A. BLONIARZ, "On the Computational Complexity of Algebra on Lattices 1," State University of New York at Albany, 1984.
23. D. KOZEN, Complexity of finitely presented algebras, *in* "Proceedings, 9th ACM Symposium on Theory of Computing, May, 1977, pp. 164–177.
24. D. MAIER, "The Theory of Relational Databases," Computer Sci. Press, Rockville, Md., 1983.
25. J. C. C. MCKINSEY, The decision problem for some classes of sentences, *J. Symbolic Logic* **8** (1943), 61–76.
26. P. PUDLAK AND J. TUMA, Every finite lattice can be embedded in a finite partition lattice. *Algebra Universalis* **10**, No. 1 (1980), 74–95.
27. R. REITER, On closed world databases, *in* "Logic and Databases" (H. Gallaire and J. Minker, Eds.), pp. 55–76, Plenum, New York, 1978.

28. F. SADRI AND J. D. ULLMAN, Template dependencies: A large class of dependencies in relational databases and its complete axiomatization, *J. Assoc. Comput. Mach.* **29**, No. 2 (1982), 363–372.
29. N. SPYRATOS, "The Partition Model: A Deductive Database Model," INRIA No. 286, April, 1984, Inst. Nat. Recherche Inform. Automat., France.
30. J. D. ULLMAN, "Principles of Database Systems," Computer Sci. Press, Rockville, Md., 1983.
31. M. Y. VARDI, personal communication, 1984.
32. Y. VASSILIOU, "A Formal Treatment of Imperfect Information in Database Management," Ph. D. thesis, University of Toronto, 1980.
33. P. M. WHITMAN, Free lattices, *An. of Math.* **42** (1941), 325–330.
34. P. M. WHITMAN, Lattices, equivalence relations and subgroups, *Bull. Amer. Math. Soc.* **52** (1946), 507–522.
35. M. YANNAKAKIS AND C. H. PAPADIMITRIOU, Algebraic dependencies, *J. Comput. System Sci.* **21** (1982), 2–41.