Algebraic Aspects
of
Relational Database Decomposition

by

Stephen J. Hegner
Department of Computer Science and Electrical Engineering
Votey Building
University of Vermont
Burlington, VT 05405

(802)656-3330

January 1983

This work was supported by the National Science Foundation under grant IST82-12178.

ABSTRACT

An algebraic framework for investigating the problem of decomposing a relational database schema into components is developed. It is argued that the views of a relational schema which are to be the components of a decomposition should form a finite atomic Boolean algebra. The unit of the algebra is the identity view, and the zero is the null view. The join operation in this algebra is to be a generalization of the usual concept of join; the resulting view should contain precisely the representation contained in the two component views as a unit. The meet operation in this algebra is to measure the interdependence of the components, and is to be zero if and only if they are independent. A decomposition of the schema is then to be a decomposition of the identity component, with the ultimate decomposition the one consisting entirely of atoms.

The thrust of the results is in two directions. First, the general properties of relational schemata particular to this problem are developed, within the framework of first-order logic. The key formulations are those of abstract meet and join of schemata. Using these formulations, it is shown that a completely general decomposition theory is impossible. The second part of the work islolates a relatively large class of schemata which do admit reasonable decompositions. These schemata include all of the usual decomposition problems, including project-join decompositions and horizontal decompositions of schemata constrained by universal Horn sentences.

0. INTRODUCTION

The problem of decomposing a relational database schema into independent components has received a great deal of attention in recent years, (*e.g.* [Riss77] [ArCa78] [BeRi80] [MMSU81] [GrYa82]). This attention is due largely to the problem's close relationship to the normalization problem [BeBG78]. These approaches have generally dealt only with single-relation schemata which are constrained by functional and/or join dependencies, and the decompositions are always taken to be vertical projections which are losslessly joinable. Recently, it has become apparent that other kinds of decompositions may be useful. For example, Smith [Smit78] and Smith and Smith [SmSm77] have argued that a horizontal type of projection is also needed in a full normalization theory. Also, recent work has suggested that other types of constraints may be useful in database theory [Fagi82] [GrJa82] [SaUl82].

It is our thesis that the understanding of the decomposition problem can be advanced significantly by examining it in a more general framework than has typically been used. We present the beginnings of such an approach in this paper. Our treatment uses two mathematical branches extensively: first-order logic and modern algebra.

Logic is used as the tool to model database schemata and mappings. Our use of logic follows the general ideas of Jacobs [Jaco79] [Jaco82] [JaAK82]; a relational database schema is a first-order theory, a database is a model of that theory, and a database mapping is an interpretation between theories. Our reasons for using logic rather than the traditional formalism are three-fold. First, logicians have developed many powerful theorems which are of direct use in our approach. Second, the notion of horizontal constraint can be readily formulated in an appropriate logical framework with a minimum of

effort; to augment the traditional framework to accomplish the same would be far more involved. Third, with logic we can talk about *all* reasonable static constraints. This is crucial in guaranteeing closure of algebraic operations.

Algebra is used as the tool to model interactions of database schemata and mappings. Our most salient applications of algebraic concepts are in the use of lattice theory to model decomposition, and in the use of continuous semilattice theory to model strong database views. However, virtually all of our formulations have an arrow theoretic flavor. While we do not explicitly use category theory as a tool, the reader having an acquaintance with the subject will certainly detect an implicit use of the basic ideas of this language.

The general outline of the paper is as follows.

In Section 1, we develop the basic properties which a good decomposition ought to have from a purely abstract point of view, using lattice theory as the foundational tool.

In Section 2, we develop the fundamental properties of database schemata and database mappings which are needed for our framework. In particular, we develop schemata defined over a *type algebra*, which is the crucial concept necessary to handle horizontal constraints and null values correctly.

In Section 3, we develop the special properties of database mappings which are necessary to our decomposition formulation. In particular, we define general concepts of what the join and meet of two arbitrary schemata are, in entirely algebraic terms. Using these definitions, we formulate the conditions necessary to decompose a relational schema. We show that a completely general decomposition theory based upon algebraic principles cannot exist.

In Section 4, we develop a decomposition theory for a large class of

schemata occurring in practice, including vertical and horizontal decomposition. The key concept used is the continuity of the view mapping with respect to model inclusion.

Section 5 contains conclusions and comments concerning the relationship of our work to that of others.

This is an extended abstract of a full paper. While we have tried to be reasonably complete with respect to definitions and statements of results, proofs are generally omitted. To guide the reader, we have terminated statements of results which need proof by an open box □, while trivial results or results which are proved here or in the literature are teminated with a solid box ■.

1. ABSTRACT DECOMPOSITION PRINCIPLES

In order to develop a viable decomposition theory, we must first have a firm idea of what it means to decompose a schema. In this section, we outline the basic algebraic principles which any decomposition must have.

Suppose that **D** is a database schema, and that the set of views $\mathbf{S} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_n\}$ is to be a decomposition of **D**. Without worrying about specific details, we may ask what properties **S** must have in order to qualify as a "good" decomposition. We claim that there are two general principles which must be adhered to: the representation principle and the independence principle.

The terminology representation principle is borrowed from [BeBG78]. Informally stated, the *representation principle* asserts that the state of **D** must be completely recoverable from the combined states of the $\mathbf{D}_i$'s. In the framework of decomposition into projections, this translates into the well-known lossless-join property [AhBU79]. Our approach postulates the existence of a general associative operator "V" on the views of **D**, which we call the *join operator*. The view $\mathbf{D}_1 \vee \mathbf{D}_2$, when it exists, is precisely that which embodies the information contained in the unification of $\mathbf{D}_1$ and $\mathbf{D}_2$. The decomposition **S** of **D** satisfies the representation principle precisely when the composite join $\mathbf{D}_1 \vee \mathbf{D}_2 \vee \ldots \vee \mathbf{D}_n$ is (isomorphic to) the original schema **D**. For example, if **D** is the single-relation schema $(\{R[ABC]\}, \{A \rightarrow B\})$, $\mathbf{D}_1 = (\{R[AB]\}, \{A \rightarrow B\})$, and $\mathbf{D}_2 = (\{R[AC]\}, \phi)$, then $\mathbf{D}_1 \vee \mathbf{D}_2$ is, (up to isomorphism), just **D**, since **D** satisfies the join dependency $\bowtie[AB,AC]$. Hence $\{\mathbf{D}_1, \mathbf{D}_2\}$ satisfies the representation principle.

The *independence principle* asserts that the components of the decomposition **S** are independent of one another. (The independence principle is related

to, but not the same as, the separation principle of [BeBG78].) Unlike the representation principle, there is no common agreement on exactly how to formulate the independence principle, even in the simple case of vertical decomposition. This is amply illustrated by the large number of papers addressing this issue, *e.g.*, [Riss77], [BeRi80], [MMSU81], [GrYa82], and [Gran82]. Our approach is again one of postulating the existence of a general operator "$\wedge$" on the views of **D**, which we call the *meet operator*. The view $\mathbf{D}_1 \wedge \mathbf{D}_2$, when it exists, is precisely that which embodies the information common to both $\mathbf{D}_1$ and $\mathbf{D}_2$; independence of the decomposition is formalized for two views by the statement $\mathbf{D}_1 \wedge \mathbf{D}_2 = \mathbf{0}$, where **0** is the empty view. For n views, it is more complicated; we shall discuss this further below. In the example of the previous paragraph, $\mathbf{D}_1 \wedge \mathbf{D}_2 = (\{R[A]\},\phi)$, since it is the A column which is common to both views. This may seem to be an incorrect formulation, since the pair $\{\mathbf{D}_1,\mathbf{D}_2\}$ is generally regarded to be a valid decomposition of **D**. However, it is definitely true that the two views are not independent; they have a domain equality constraint, in the sense of Kent [Kent81]. We shall see that the resolution of this apparent inconsistency rests in the appropriate *formalization* and use of null values in the definition of the schema **D**; with the proper use of nulls, we can achieve true independence.

We now turn to a mathematical formulation of these independence concepts within the framework of lattice theory and Boolean algebras. We outline only the concepts essential to our approach. For details, consult the texts of Grätzer [Grat78], Birkhoff [Birk67], and Halmos [Halm74].

A *lattice* **L** is a set L with a partial ordering $\leq$ for which each pair of elements (x,y) has a least upper bound (lub) $x \vee y$ and a greatest lower bound (glb) $x \wedge y$. We also apply these operators to sets of elements: $\vee X$, $\wedge X$. The operation $\vee$ is called the *join* of the lattice while the operation $\wedge$ is called

the *meet*. We usually represent **L** as an algebra $(L,\vee,\wedge)$ with these operations, as they completely determine $\leq$. **L** is *distributive* if it satisfies the distributive law: $x\wedge(y\vee z) = (x\wedge y)\vee(x\wedge z)$. **L** has *universal bounds* if it has a largest element (which we denote by **1**) and a smallest element (which we denote by **0**). x and y are *complementary* (or y is a *complement* of x) if $x\vee y = \mathbf{1}$ and $x\wedge y = \mathbf{0}$. x is an *atom* if $\mathbf{0} \leq y \leq x \implies y = x$ or $y = 0$. *For the rest of this section, assume that* **L** = $(L,\vee,\wedge)$ *is a lattice with universal bounds.*

1.1 DEFINITION (a) A *decomposition* is a subset X of $L\setminus\{\mathbf{0}\}$ such that:

(i) $\vee X = \mathbf{1}$;

(ii) for any partition $\{X_1,X_2\}$ of X, $(\vee X_1)\wedge(\vee X_2) = \mathbf{0}$.

(b) If X and Y are decompositions, Y is a *refinement* of X if every element of Y is a join of elements of X.

(c) A decomposition X is *maximal* if for every refinement Y of X, Y = X. X is *ultimate* if it is a refinement of every decomposition.

In algebraic terms, our decomposition problem can be translated into the study of lattices of views of **D** in which the top element is the schema to be decomposed, the bottom element is the empty view, and join and meet define the properties outlined above. Ideally, we would like each schema to have an ultimate decomposition, so that we can achieve a decomposition into "atomic components" [Riss77]. We now turn to characterizing properties of the lattice which will ensure this.

1.2 FACTS *Let* L *be a distributive lattice.*

(a) *If* x *has a complement, then it is unique. We denote this unique complement by* $\bar{x}$. *Thus, we cannot have two distinct decompositions* $\{x_1,x_2\}$ *and*

$\{x_1, x_3\}$ *with* $x_2 \neq x_3$.

(b) *Any two decompositions have a common refinement. Therefore, maximal decompositions are ultimate.*

(b) 1.1(a)(ii) *can be weakened to:* For any $x_i, x_j \in X$, $x_i \wedge x_j = \mathbf{0}$. *In other words, pairwise independence is sufficient to guarantee global independence.*

(c) *The set of all elements of* L *which have complements forms a distributive sublattice of* **L**, *which we denote by* Comp(**L**). ∎

A distributive lattice with unique complements is called a *Boolean lattice*; if we make the complementation operation explicit: $\mathbf{L} = (L, \vee, \wedge, ^-)$, then **L** is called a Boolean algebra. Thus, if **L** is distributive, Comp(**L**) may be regarded as a Boolean algebra. We call Comp(**L**) the *component algebra* of **L**. Our main tool is the following.

1.3 MAIN DECOMPOSITION TOOL *Let* **L** *be a distributive lattice with the property that its component algebra is finite. Then the atoms of the component algebra form an ultimate decomposition.* ∎

Thus, our goal shall be to isolate schemata and classes of views which admit the structure of a distributive lattice whose component algebra is finite.

2. SCHEMATA AND MAPPINGS

We presume a basic familiarity with first-order mathematical logic, such as can be found in [Ende72] or [Monk76]. For purposes of establishing notation, we recall some basic terminology. The logical connectives we use are $(,),,,\vee,\wedge,\neg,\Longrightarrow$, and $\Longleftrightarrow$. We also have at our disposal a countable set of variables $(x_1,x_2,\ldots)$. We sometimes use x,y,z,w, etc. as variables. A *first-order language with equality* **L** consists of these logical connectives and variables, a set **R** of relation symbols, each with finite arity, including the equality relation symbol =, and set of constant symbols **K**. We do not consider languages with non-nullary functional symbols in this work. A *structure* (or *interpretation*) M for **L** consists of a set S, together with the assignment of any n-ary relation R^M to each relational symbol of arity n, and a single element $a^M \in S$ to each constant symbol a. An *axiom* for **L** is any sentence (= well-formed formula with no free variables) in the language of **L**. If A is a set of axioms, we say that M is a *model* of A if every axiom of A is true for M. If w is any sentence in the language of **L**, we write $A \models w$ if w is true for every model of A. The set $\{w \mid A \models w\}$ is called the *theory* of A, and is denoted by A^+. A is a *theory* if $A = A^+$.

Our basic philosophy regarding the use of logical concepts in database theory follows the general lines of the approach of Jacobs [Jaco79] [Jaco82] [JaAK82]. Namely, a (relational) database schema is a first-order theory, and a database over this theory is just a model of its axioms. Our framework differs from that presented in [JaAK82] in the following ways:

(a) We do not allow (non-nullary) function symbols in the language. Their presence, within the context of the problems we address, causes some problems. Furthermore, nothing corresponding to them is present in the more traditional

approach to the subject.

(b) We work with a Boolean algebra of types, rather than just a set of types. This idea, which has also been used in [McMi79] and [Reit80], will allow us to treat "horizontal" dependencies and null values in a formal fashion.

(c) The types and constant symbols of the language are fixed within a given framework; all schemata within a given context use the same types and constant symbols, with the same meaning. Database mappings preserve these types and constants identically.

2.1 Type Algebras

In the traditional approach to relational database theory [Ullm80], we typically start with a set of *attributes*. With each attribute A we then associate a *domain* dom(A). Attributes become labels for columns of relation symbols, with actual relations consisting of tuples from the appropriate domains. In the logic-based approach of [JaAK82], attributes amount to *types* in a many-sorted theory, with domains just the domains of interpretation. In either case, the attributes are taken to be totally independent; the domains are pairwise disjoint. Thus, we cannot directly consider any kind of constraint which has a "horizontal" component in the sense that the type of entry in a tuple is important. The work of Smith [Smit78] and Smith and Smith [SmSm77] has shown the importance of such constraints in database decomposition. We therefore seek a framework in which such ideas can easily be represented.

The idea of using a Boolean algebra of types, rather than just a set of types, can be found in the work of McSkimin and Minker [McMi79] and in the work of Reiter [Reit80]. Ours is a minor variation of these; in particular, we follow the ideas of Reiter closely.

2.1.1 DEFINITION A *type algebra* is a triple $T = (\mathbf{T},\mathbf{K},\mathbf{A})$, where:

(a) **T** is a finite set of unary predicate symbols, called the *types*,

(b) **K** is an at most countable set of constant symbols, called the *names*.

(c) **A** is a set of axioms in the first-order language with equality whose relation symbols are precisely those of **T** and whose constant signs are precisely those of **K**.

(d) For each τ_1, $\tau_2 \in \mathbf{T}$, there is a $\tau_3 \in \mathbf{T}$ such that $\mathbf{A} \models (\forall x)(\tau_3(x) \iff \tau_1(x) \vee \tau_2(x))$. We denote τ_3 by $\tau_1 \vee \tau_2$ and call it the *union* of τ_1 and τ_2.

(e) For each τ_1, $\tau_2 \in \mathbf{T}$, there is a $\tau_3 \in \mathbf{T}$ such that $\mathbf{A} \models (\forall x)(\tau_3(x) \iff \tau_1(x) \wedge \tau_2(x))$. We denote τ_3 by $\tau_1 \wedge \tau_2$ and call it the *intersection* of τ_1 and τ_2.

(f) For each $\tau \in \mathbf{T}$, there is a $\bar{\tau} \in \mathbf{T}$ such that $\mathbf{A} \models (\forall x)(\bar{\tau}(x) \iff \neg\tau(x))$. We call $\bar{\tau}$ the *complement* of τ.

(g) There is a type $\tau_u \in \mathbf{T}$ such that $\mathbf{A} \models (\forall x)(\tau_u(x))$. We call τ_u the *universal type*.

(h) For any $k \in \mathbf{K}$ and any $\tau \in \mathbf{T}$, we have either $\mathbf{A} \models \tau(k)$ or $\mathbf{A} \models \neg\tau(k)$.

The types correspond exactly to the attributes of the usual approach. It is clear that conditions (d)-(g) endow **T** with the structure of a finite Boolean algebra. In particular, τ_u is the 1 element of the algebra; its complement is the empty type τ_ϕ. Condition (h) says that the type of any constant is completely determined by **A**. This is the condition of τ-completeness of [Reit80].

The domains of these types are defined via a logical interpretation. More precisely, a *type assignment* for T is any model μ of **A**; *i.e.*; a set X together with an assignment of a subset of X to each type symbol τ and an

assignment of an element of X to each constant symbol k such the axioms in **A** are satisfied. Note that under this setup, X truly becomes a Boolean algebra under the usual set-theoretic operations of union, intersection, and complement.

As a notational convenience, throughout the rest of this section, we fix a type algebra $T = (\mathbf{T},\mathbf{K},\mathbf{A})$.

2.2 Database Schemata

2.2.1 DEFINITION A *database schema over* T is a pair $\mathbf{D} = (\mathbf{R},\mathbf{C})$, where
(a) **R** is a finite set of relation symbols, with $\mathbf{R} \cap \mathbf{T} = \phi$.
(b) **C** is the set of *constraints* of **D**. It is just a set of axioms in the *language of* **D**, which is the language with equality whose other relation symbols are those of $\mathbf{R} \cup \mathbf{T}$, and with no constant symbols. Thus, we do *not* let constraints talk about individual named objects.

Note that we allow the types to enter into the constraint definitions. For example, suppose we have a ternary relation symbol R whose domains are A, B, and C respectively. In the traditional approach we would write R[ABC]. In our approach, the domain constraint would be an axiom. If τ_A, τ_B, and τ_C, are the types corresponding to these domains, we would express this typing constraint as

$$(\forall x)(\forall y)(\forall z)(R(x,y,z) \Longrightarrow \tau_A(x) \wedge \tau_B(y) \wedge \tau_C(z)).$$

Of course, we still can (and will) write R[ABC], with the understanding that the above sentence is what this abbreviation means. We shall also use the common abbreviations for functional and join dependencies. For example, we regard $A \rightarrow B$ as an abbreviation for the axiom

$(\forall x)(\forall y)(\forall z)(\forall u)(\forall w)(R(x,y,z)\wedge R(x,u,w) \Longrightarrow u=y)$,

and we regard $\bowtie[AB,BC]$ as an abbreviation for the axiom

$(\forall x)(\forall u)(\forall y)(\forall z)(\forall w)(R(u,y,z)\wedge R(x,y,w) \Longrightarrow R(u,y,w))$.

It is in fact well-known that all of the commonly considered constraints can be expressed as sentences in the first-order language of the schema. See, *e.g.*, [Fagi82] and [GrJa82]. Our framework also admits and elegant representation of horizontally embedded constraints.

2.2.2 EXAMPLE As a concrete example, consider the schema consisting of a single binary relation symbol R with the typing constraint R[AB]. The domain A represents employees, and the domain B represents projects, with R(x,y) meaning that employee x works on project y. Suppose that projects are further broken down into two categories, classified and unclassified. The constraint is that an employee can work on any number of unclassifed projects, but he can work on at most one classified project. We thus have a horizontally embedded functional dependency, in the sense of [Smit78]. This is easily represented within our framework. Let τ_e, τ_{nc}, τ_c, and τ_p denote the types for employees, unclassified projects, classified projects, and all projects. Note that we have $\tau_A = \tau_e$, $\tau_B = t_{nc}\vee\tau_c = \tau_p$ and $t_{nc}\wedge\tau_c = \tau_\phi$. The main typing constraint for the relation is

$$(\forall x)(\forall y)(R(x,y) \Longrightarrow \tau_e(x)\wedge\tau_p(y)).$$

The embedded functional dependency is given by

$$(\forall x)(\forall y)(\forall z)(R(y,x)\wedge R(y,z)\wedge\tau_c(x)\wedge\tau(z) \Longrightarrow x=z)).$$

The schema can be normalized by decomposing it horizontally into the two binary relations S and T defined by

$$(\forall x)(\forall y)(S(x,y) \Longleftrightarrow R(x,y)\wedge\tau_{nc}(y)) \quad \text{and}$$

$$(\forall x)(\forall y)(T(x,y) \Longleftrightarrow R(x,y)\wedge\tau_c(y)).$$

We shall formalize the notion of horizontal decomposition more completely in Section 4. We note that Grant [Gran82] has formalized such constraints with the similar notion of *condition declaration.*

2.2.3 EXAMPLE Within this framework, we view null values as special types with exactly one element. For example, consider a relational schema with a single ternary relation symbol R with typing constraint R[ABC]. Suppose further that R is to satisfy the join dependency ⋈[AB,BC]. This is all easily represented, as illustrated above. However, suppose that we wish to allow components in the relation which only involve AB or only involve BC. In the terminology of Sciore [Scio80], we wish these to be objects. To do this, we augment the type structure as follows. In addition to the basic domain types τ_A, τ_B, and τ_C, we add a new type, τ_n. This type is constrained to have just one element; for simplicity, let us take it to be a constant n. Thus, as part of the type algebra axioms, we have

$\tau_n(n) \wedge (\forall x)(\tau_n(x) \Longrightarrow x=n)$.

The type constraint on the relation becomes

$(\forall x)(\forall y)(\forall z)(R(x,y,z) \Longrightarrow (\tau_A \vee \tau_n)(x) \wedge \tau_B(y) \wedge (\tau_C \vee \tau_n)(z))$.

In other words, we allow the null value in either column 1 or column 3. We do not want to allow the null value in both columns, so we impose

$(\forall x)(\forall y)(\forall z)(R(x,y,z) \Longrightarrow (\neg\tau_n(x) \vee \neg\tau_n(z)))$.

Now the join dependency, modified for null values, says that triples without null values should be joined. In other words, the join dependency becomes

$(\forall x)(\forall u)(\forall y)(\forall z)(\forall w)(R(u,y,z) \wedge R(x,y,w) \wedge \tau_A(u) \wedge \tau_C(z) \wedge \tau_A(x) \wedge \tau_C(w) \Longrightarrow R(u,y,w))$.

As we shall see, this is the "true" join dependency which we want for decomposition. This modified schema is almost what we want for a decomposition. For further discussion of how to decompose it, see 3.2.4, 3.4.2, 3.5.2, 4.1.12,

4.1.13 and 4.2.10.

The key point here is the formalization of the null values. Note that they are incorporated right into the language of dependency description. While we *could* do this with ordinary many-sorted logic, it would be clumsier, since we would constantly have to assert that the null η could not be used as an ordinary domain element. Let us remark that we are not giving any particular semantics to null values at this point. Whatever interpretation the schema designer wishes to assign to them, consistent with the semantics of first-order logic, is acceptable.

2.3 Database Instances

2.3.1 DEFINITION Let $\mathbf{D} = (\mathbf{R},\mathbf{C})$ be a database schema over τ, and let μ be a type assignment for τ.

(a) A *database* (or *structure*, or *instance*) of $\mathbf{D}$, with respect to μ, is just an interpretation M of the language of $\mathbf{D}$ which is a model of $\mathbf{A}$, but not necessarily of $\mathbf{C}$. In other words, a database of $\mathbf{D}$ is any structure of the schema $\mathbf{D}$ which is consistent with μ. The set of all databases of $\mathbf{D}$, with respect to μ is denoted by $DB(\mathbf{D},\mu)$.

(b) A database is *legal* if it is a model of $\mathbf{C}$. In other words, legal databases are those which satisfy the constraints of the schema. The set of all legal databases of $\mathbf{D}$, with respect to μ, is denoted by $LDB(\mathbf{D},\mu)$.

Note that we do not restrict consideration to finite databases. This is done for mathematical reasons. It is an unfortunate fact that some of the most useful results of first-order logic (*e.g.* Beth's theorem [Monk76, Thm. 22.4]) fail when attention is restricted to finite structures. We are there-

fore forced to consider the more general case.

It is also important to observe that the notion of database is with respect to a particular type assignment μ. Intuitively, the idea is that we first fix a type assignment. We then consider database concepts with respect to this particular assignment. While it might seem more reasonable to fix μ once and for all, we again run into mathematical problems if we do this; in order to use key results from logic, we must be able to consider all possible models of a theory, not just those over a fixed domain. However, *to avoid saying "for all type assignments* μ" *in every definition and theorem, whenever a type assignment* μ *is involved in a statement, it shall implicitly be assumed that* μ *is to be allowed to range over all possible type assignments.*

2.4 Database Mappings

Given two database schemata, a database mapping is a rule which maps structures of the first schema into structures of the second. The idea of using interpretations between theories to model database mappings is due to Jacobs [Jaco79] [JaAK82]. The key difference betweeen our definition and his is that we only consider mappings between schemata over the same type algebra, and we require that the mappings be the identity on that algebra.

2.4.1 DEFINITION Let $\mathbf{D}_1 = (\mathbf{R}_1, \mathbf{C}_1)$ and $\mathbf{D}_2 = (\mathbf{R}_2, \mathbf{C}_2)$ be database schemata. A *database mapping* from $\mathbf{D}_1$ to $\mathbf{D}_2$ is an interpretation f of the language of $\mathbf{D}_2$ into the language of $\mathbf{D}_1$, *which is the identity on* $\mathbf{T} \cup \mathbf{K}$, and such that the interpretation formula for each $R \in \mathbf{R}_2$ does not contain any constant symbols. We write $f : \mathbf{D}_1 \rightarrow \mathbf{D}_2$.

Under our definition, f is completely specified by giving, for each n-ary $R \in \mathbf{R}_2$, a formula ψ_R^f, in the language of $\mathbf{D}_1$ with exactly n free variables and no constant symbols. R is then defined by

$$R(x_1,x_2,\ldots,x_n) \iff \psi_R^f(x_1,x_2,\ldots,x_n).$$

None of the other aspects of defining an interpretation, as given in [Jaco79] or [JaAK82], is necessary, since both the type algebra and defining domains are left untouched by the transformation. The defining formula induces a mapping of structures $f_\mu^* : LDB(\mathbf{D}_1,\mu) \rightarrow LDB(\mathbf{D}_2,\mu)$ for each type assignment μ in the obvious fashion.

2.4.2 DEFINITIONS Let $f : \mathbf{D}_1 \rightarrow \mathbf{D}_2$ be a database mapping.

(a) f is *legal* or *correct* if f_μ^* maps $LDB(\mathbf{D}_1,\mu)$ into $LDB(\mathbf{D}_2,\mu)$ for each type assignment μ. In this case, we denote the restriction by $f_\mu' : LDB(\mathbf{D}_1,\mu) \rightarrow LDB(\mathbf{D}_2,\mu)$. A legal database mapping is also called a *database morphism*. When no confusion can result, we shall often write f' in lieu of the more clumsy f_μ'. Similarly, we may write f^* for f_μ^*.

(b) Relative to f, an *implied constraint* of $\mathbf{D}_2$ is a sentence in the language of $\mathbf{D}_2$ which is satisfied by all structures of $\mathbf{D}_2$ of the form $f_\mu^*(M)$, where $M \in LDB(\mathbf{D}_1,\mu)$. We clearly have that f is correct if and only if each $C \in \mathbf{C}_2$ is an implied constraint. For an elegant solution to the implied constraint problem using logic, see [JaAK82].

(c) f is *faithful* if whenever C is an implied constraint of $\mathbf{D}_2$, then $\mathbf{C}_2 \models C$. In other words, f is faithful if $\mathbf{D}_2$ carries the largest set of axioms possible while still maintaining correctness of the mapping.

(d) f is *surjective* if it is correct and the induced mapping $f_\mu' : LDB(\mathbf{D}_1,\mu) \rightarrow LDB(\mathbf{D}_2,\mu)$ is surjective for each type assignment μ.

It is easy to see that every surjective database mapping is faithful, but the converse is, unfortunately, not true. We illustrate with an example.

2.4.3 EXAMPLE Let $\mathbf{D}_1$ be the database schema with the single binary relation symbol R[AB] and the functional dependency constraints {A → B , B → A}. Let $\mathbf{D}_2$ be the database schema with two unary relation symbols S[A] and T[B]. Consider the database mapping $f : \mathbf{D}_1 \rightarrow \mathbf{D}_2$ which is just the projections of the appropriate columns. That is,

$S(x) \iff (\exists y)(R(x,y))$

$T(x) \iff (\exists y)(R(y,x))$.

The two functional dependencies of $\mathbf{D}_1$ place the elements of the two columns of R into bijective correspondence. The images of such relations under f are precisely the pairs of relations which have exactly the same number of elements. That is, if #S (resp. #T) denotes the number of elements in the instance of S (resp. T), then the constraint required for f to be surjective is that #S = #T. However, it easily seen that this cannot be a first-order constraint. For example, if #S is countably infinite and #T is uncountably infinite, there is no first-order sentence which can determine that they are not of the same size.

It is true that every faithful database mapping is surjective on *finite* models; that is, if f is faithful and M is a legal database of $\mathbf{D}_2$ which has only a finite number of tuples, then there is a legal database N of $\mathbf{D}_1$ such that $f'(N) = M$.

2.4.4 DEFINITION Let $f : \mathbf{D}_1 \rightarrow \mathbf{D}_2$ be a legal database mapping. The *faithful closure* of $\mathbf{D}_2$ with respect to f is the pair $(\mathbf{R}_2, \mathbf{C}_2^f)$, where $\mathbf{C}_2^f$ is the set of all axioms which are implied constraints of $\mathbf{D}_2$ relative to f. The *faithful*

closure of f is just the interpretation of the faithful closure of $\mathbf{D}_2$ in $\mathbf{D}_1$ via f.

2.4.5 DEFINITION Let $f : \mathbf{D}_1 \to \mathbf{D}_2$ and $g : \mathbf{D}_2 \to \mathbf{D}_3$ be database mappings. We may form the composition g∘f in an unambiguous way. Namely, for each relation symbol R in the relation name set $\mathbf{R}_3$ of $\mathbf{D}_3$, we have an interpretation formula

$$R(x_1,x_2,\ldots,x_n) \iff \psi_R^g(x_1,x_2,\ldots,x_n)$$

into the language of $\mathbf{D}_2$. To get the composition, replace each occurrence of each $R' \in \mathbf{R}_2$ in ψ_R^g with its defining formula in the language of $\mathbf{D}_1$. The result is the defining formula for R in the language of $\mathbf{D}_1$.

It is easy to see that $(g\circ f)^* = g^*\circ f^*$, and, if both f and g are morphisms, $(g\circ f)' = g'\circ f'$. Furthermore, if both f and g are faithful (resp. surjective), so too is the composition.

2.4.6 DEFINITIONS The identity morphism $\mathbf{1} : \mathbf{D}_1 \to \mathbf{D}_1$ is defined in the obvious way. The database morphism $f : \mathbf{D}_1 \to \mathbf{D}_2$ is an *isomorphism* if there is a database morphism $g : \mathbf{D}_2 \to \mathbf{D}_1$ such that both g∘f and f∘g are identity morphisms. We say that $\mathbf{D}_1$ and $\mathbf{D}_2$ are *isomorphic* if there is an isomorphism $f : \mathbf{D}_1 \to \mathbf{D}_2$.

3. DATABASE VIEWS

Throughout this section, we fix a type algebra $T = (\mathbf{T},\mathbf{K},\mathbf{A})$. All database schemata are assumed to be taken over this algebra. We also fix a database schema $\mathbf{D} = (\mathbf{R},\mathbf{C})$.

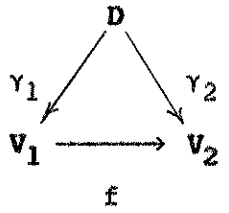
3.1 Basic Concepts of Views

If a schema $\mathbf{V}$ is to qualify as a view of $\mathbf{D}$, it must be the case that the state of $\mathbf{D}$ completely determines the state of $\mathbf{V}$. This in turn means that there must be a specified *surjective* database morphism from $\mathbf{D}$ to $\mathbf{V}$. We formalize this as follows.

3.1.1 DEFINITION A *view* of $\mathbf{D}$ is a pair $\Gamma = (\mathbf{V},\gamma)$, where $\mathbf{V}$ is a database schema and $\gamma : \mathbf{D} \rightarrow \mathbf{V}$ is a surjective database morphism. We sometimes refer to the mapping γ as the view.

It will be extremely important for us to be able to allow views to interact with each other. As a first step, we define what it means to have a morphism from one view to another.

3.1.2 DEFINITION Let $\Gamma_1 = (\mathbf{V_1},\gamma_1)$ and $\Gamma_2 = (\mathbf{V_1},\gamma_1)$ be views of $\mathbf{D}$. A *morphism* from Γ_1 to Γ_2 is a surjective database morphism $f : \mathbf{V_1} \rightarrow \mathbf{V_2}$ such that the following diagram commutes.



We write $f : \Gamma_1 \to \Gamma_2$. View morphisms compose in the obvious way. We let View(**D**) denote the class (actually the category) of all views over **D**.

(b) Γ_1 and Γ_2 are *isomorphic* if they satisfy the usual categorical notion of isomorphism: there are morphisms $f : \Gamma_1 \to \Gamma_2$ and $g : \Gamma_2 \to \Gamma_1$ such that $f \circ g$ and $g \circ f$ are identities.

A morphism from one view to another is very special; there can be at most one. This induces a partial ordering on the equivalence classes of isomorphic views, as we now illustrate.

3.1.3 PROPOSITION *Let* Γ_1 *and* Γ_2 *be views of* **D**. (a) *There is at most one morphism* $f : \Gamma_1 \to \Gamma_2$, *and* f *must necessarily be surjective.*

(b) Γ_1 *and* Γ_2 *are isomorphic if and only if there are view morphisms* $f : \Gamma_1 \to \Gamma_2$ *and* $g : \Gamma_2 \to \Gamma_1$.

Proof: (a) The function $\gamma_1' : \mathrm{LDB}(\mathbf{D},\mu) \to \mathrm{LDB}(\mathbf{V_1},\mu)$ is surjective, by the definition of view. Hence there can be at most one function $h : \mathrm{LDB}(\mathbf{V_1},\mu) \to \mathrm{LDB}(\mathbf{V_2},\mu)$ such that $\gamma_1' = \gamma_2' \circ h$. Thus there can be at most one database morphism f making the above diagram commute. It clearly must be surjective.

(b) is immediate. ∎

3.1.4 DEFINITION Let QView(**D**) denote the equivalence classes of views of **D**

defined by the equivalence relation $\Gamma_1 \equiv \Gamma_2$ if and only if Γ_1 and Γ_2 are isomorphic. Define the relation $\leq$ on QView(**D**) by $X \leq Y$ if and only if there is a view morphism $f : \Gamma_1 \rightarrow \Gamma_2$ for some $\Gamma_1 \in Y$ and $\Gamma_2 \in X$.

While View(**D**) will be a proper class and not a set, in general, QView(**D**) is a set. To see this, note that there are only a countable number of schemata, up to isomorphism (since each has only a finite number of relation symbols), and that there are an at most countable number of database mappings between schemata (since a database mapping is defined by a finite number of formulas over an at most countable language).

Throughout the rest of this section, we shall not distinguish between members of QView(**D**) and individual views. In other words, we do not distinguish between isomorphic views. When we speak of a view in QView(**D**), we mean any representative of an equivalence class.

The following is an immediate consequence of 3.1.4.

3.1.5 PROPOSITION *$\leq$ is a partial ordering of the set* Qview(**D**). ■

3.1.6 DEFINITION Let μ be a type assignment for T, and let $\Gamma = (\mathbf{V},\gamma)$ be a view of **D**. The mapping f'_μ : LDB(**D**,μ) $\rightarrow$ LDB(**D**,μ) defines an equivalence relation on LDB(**D**,μ) by $M \equiv N$ if $f'_\mu(M) = f'_\mu(N)$. We denote the partition defined by this equivalence relation by π^f_μ.

3.1.7 PROPOSITION *Given any two views* $\Gamma_1 = (\mathbf{V_1},\gamma_1)$ *and* $\Gamma_2 = (\mathbf{V_2},\gamma_2)$ *of* **D**, $\Gamma_1 \leq \Gamma_2$ *if and only if* $\pi^\gamma_\mu 1$ *is coarser than* $\pi^\gamma_\mu 2$ for every type assignment μ. □

This proposition gives a link with the work of Bancilhon and Spyratos

[BaSp81]. They consider a database schema to be just a set S. A database mapping in their framework is essentially any function f with domain S. They then investigate properties of such functions by investigating the equivalence relations which they induce on S, in the same way that our interpretations induce the equivalence relations Π_μ. Our investigation differs from theirs primarily in that while they admit any function as a database mapping, we consider only those which arise from logical interpretations.

This partial ordering represents the order which we will use in our quest to obtain a natural decomposition of **D** within a lattice framework. As a first step, let is note that this ordering has universal bounds.

3.1.8 DEFINITION The *identity view* is $(\mathbf{D},\mathbf{1_D})$. It is denoted by $\mathbf{1_D}$, or by just **1** if the context **D** is clear.
(c) The *zero view* is $(\mathbf{nil},\mathbf{0_D})$; it is denoted by $\mathbf{0_D}$, or by just **0** if the context **D** is clear.

3.1.9 OBSERVATION *The partial ordering* $\leq$ *of* QView(**D**) *has universal bounds; the identity view is the universal upper bound, and the zero view is the universal lower bound.* ∎

For the rest of this section, let $\Gamma_1 = (\mathbf{V_1},\gamma_2)$ *and* $\Gamma_2 = (\mathbf{V_2},\gamma_2)$ *be (equivalence classes of) views of* **D**, *with* $\mathbf{V_1} = (\mathbf{R_1},\mathbf{C_2})$ *and* $\mathbf{V_2} = (\mathbf{R_2},\mathbf{C_2})$. In further selecting representatives, we shall assume that $\mathbf{R_1} \cap \mathbf{R_2} = \phi$. This convention is in no way critical; it will merely simplify the notation of results which combine the two views. Alternatively, we may think of selecting Γ_1 and Γ_2, and *then* renaming the relation names to insure that there are none in common.

3.2 Joins of Views

We now turn to the problem of constructing a join of two views. In doing so, we must be careful to make sure that our concept of join really does what we want in terms of decomposition, rather than just being a lub of two objects in the ordering. Thus, we start with a decompositiion-oriented definition. The join Γ of the two views Γ_1 and Γ_2 must have two properties:

(i) It must be greater than both Γ_1 and Γ_2.

(ii) It must be defined entirely by Γ_1 and Γ_2.

The first property says that it is big enough; this is guaranteed if we respect the natural ordering of views. The second says that it is not too big, and is more subtle. There is nothing in the structure of this ordering that would warrant, *a priori*, a conclusion that any supremum would satisfy (ii). Nonetheless, we certainly want (ii) to hold, else a decomposition might violate the representation principle. We are therefore forced to make a more concrete definition of join, and then show that it satisfies the requirement that it be a supremem in the ordering. The following definition formalizes condition (ii) above.

3.2.1 DEFINITION Let $\Gamma = (\mathbf{V},\gamma)$ be a view of **D**. Γ is *implicitly definable* by $\{\Gamma_1,\Gamma_2\}$ if for any two $M, N \in LDB(\mathbf{D},\mu)$, whenever $\gamma_1^!(M) = \gamma_1^!(N)$ and $\gamma_2^!(M) = \gamma_2^!(N)$, it must be that $\gamma'(M) = \gamma'(N)$ also.

3.2.2 DEFINITION Let $\Gamma = (\mathbf{V},\gamma)$ be a view of **D**. Γ is a *join* of Γ_1 and Γ_2 if it satisfies the following two conditions.

(i) $\Gamma_1 \leq \Gamma$ and $\Gamma_2 \leq \Gamma$.

(ii) Γ is implicitly definable by $\langle \Gamma_1, \Gamma_2 \rangle$.

The above definition tells us nothing about how to build the join. Fortunately, there is a simple construction for the join, when it exists.

3.2.3 DEFINITION (a) The *free product* of Γ_1 and Γ_2, denoted $\Gamma_1 \times \Gamma_2$, is the pair $(\mathbf{V_1} \times \mathbf{V_2}, \gamma_1 \times \gamma_2)$, with $\mathbf{V_1} \times \mathbf{V_2}$ the schema given by $(\mathbf{R_1} \cup \mathbf{R_2}, \mathbf{C_1} \cup \mathbf{C_2})$, and with $\gamma_1 \times \gamma_2 : \mathbf{D} \rightarrow \mathbf{V_1} \times \mathbf{V_2}$ the database morphism defined for each $R \in \mathbf{R_1} \cup \mathbf{R_2}$ by

$R(x_1, x_2, \ldots, x_n) \iff \psi_R^{\gamma 1}(x_1, x_2, \ldots, x_n)$ if $R \in \mathbf{R_1}$;

$R(x_1, x_2, \ldots, x_n) \iff \psi_R^{\gamma 2}(x_1, x_2, \ldots, x_n)$ if $R \in \mathbf{R_2}$.

Since $\mathbf{R_1}$ and $\mathbf{R_2}$ are assumed to be disjoint, this definition is totally unambiguous.

(b) The *constrained product* of Γ_1 and Γ_2, denoted $\Gamma_1 \circledast \Gamma_2$, is the pair $(\mathbf{V_1} \circledast \mathbf{V_2}, \gamma_1 \times \gamma_2)$, where $\mathbf{V_1} \circledast \mathbf{V_2}$ is the schema with the same relation symbols as $\mathbf{V_1} \times \mathbf{V_2}$, but whose axioms are defined as those of the faithful closure of $\gamma_1 \times \gamma_2$.

The free product is just the "union" of the two views, with no interrelational constraints. The constrained product is the union with the interrelational constraints. We illustrate with a few examples.

3.2.4 EXAMPLE Let **D** be the initial single-relation schema of Example 2.2.3: R[ABC] with the single join dependency $\bowtie$[AB,BC]. Let $\Gamma_1 = (\langle\{R_{12}\}, \phi\rangle, \pi_{12})$ be the AB projection of R. Similarly, let $\Gamma_2 = (\langle\{R_{23}\}, \phi\rangle, \pi_{23})$ be the BC projection of R. Then $\Gamma_1 \times \Gamma_2$ is just the pair $(\langle\{R_{12}, R_{23}\}, \phi\rangle, \pi_{12} \times \pi_{23})$. There are no interrelational constraints at all; any two relations on the appropriate domains are legal. On the other hand, $\Gamma_1 \circledast \Gamma_2$ is the pair $(\langle\{R_{12}, R_{23}\}, \{R_{12}[B] \subseteq R_{23}[B], R_{23}[B] \subseteq R_{12}[B]\}\rangle, \pi_{12} \times \pi_{23})$, where the constraints

$R_{12}[B] \subseteq R_{23}[B]$ and $R_{23}[B] \subseteq R_{12}[B]$ are shorthand for the sentences

$(\forall x)(\forall y)(\exists z)(R_{12}(x,y) \Longrightarrow R_{23}(y,z))$, and

$(\forall x)(\forall y)(\exists z)(R_{23}(x,y) \Longrightarrow R_{12}(z,x))$ repectively.

These are the so-called *inclusion dependencies*, as discussed in [CaFP82]. This is just the property necessary for the two relations to be joinable in lossless fashion. We note that in this case the constrained product is a view, but the free product is not, since it is clearly not surjective.

3.2.5 THEOREM (a) *The join of* Γ_1 *and* Γ_2, *when it exists, is unique, up to isomorphism. It exists if and only if the constrained product* $\Gamma_1 \circledast \Gamma_2$ *is a view (i.e., is surjective) and in this case is given by this product.* We also denote the join by $\Gamma_1 \vee \Gamma_2$.

(b) *The join operation satisfies the following limited associativity condition: If* Γ_1, Γ_2, *and* Γ_3 *are views which are pairwise joinable, then both* $(\Gamma_1 \vee \Gamma_2) \vee \Gamma_3$ *and* $\Gamma_1 \vee (\Gamma_2 \vee \Gamma_3)$ *exist and are isomorphic.* □

3.2.6 EXAMPLE The example of 3.2.4 is a join. It is easily seen that the resulting constrained product is isomorphic to the original schema. Therefore, this general concept of join coincides with the usual one.

3.2.7 EXAMPLE Consider again the example of horizontal decomposition of Example 2.2.2. The views we consider are $\Gamma_1 = (\{\{S\},\{S[e,nc]\}\},\sigma_{2,nc})$ and $\Gamma_2 = (\{\{T\},\{T[e,c],e \rightarrow c\}\},\sigma_{2,c})$. Here $\sigma_{2,nc}$ is the restriction (= horizontal projection) of the original relation R to those tuples whose column 2 entry is of type τ_{nc}. Similarly, $\sigma_{2,c}$ is the restriction of those tuples whose column 2 entry is of type τ_c. A join of these two schemata is just the free product, as there are no interrelational constraints.

3.2.8 EXAMPLE It is unfortunately not the case that the join of two views always exists. As a simple example, consider again Example 2.4.3. It is clear that $f = \pi_1 \times \pi_2$, where the π's are the obvious projections. As argued in 2.4.3, the faithful closure of f is not surjective. Thus, the join of these two views cannot exist.

Let us also remark that the identity view $\mathbf{1}_{\mathbf{D}}$ is in fact the supremum of these two projections in the natural ordering defined in 3.1.4. Thus, we cannot use this ordering as an equivalent definition of join.

We can relate our join to a similar concept in the work of [BaSp81] with the following proposition.

3.2.9 PROPOSITION *If the join of* Γ_1 *and* Γ_2 *exists, then for any type assignment* μ, *the induced partition on* LDB(**D**,μ) *is the supremum of the partitions induced by* Γ_1 *and by* Γ_2. □

Thus, our concept of join corresponds to their notion of the supremum of two views. Unlike our case, suprema always exist in their framework. The reason is that they work with just sets, rather than logical theories. This is more a matter of what is acceptable as a view, rather than matter of relative strength of approaches. In our framework, it is clear what the join must be (consider, *e.g.* Example 3.2.8 above); it is just that it is not axiomatizable with first-order sentences.

3.3 Meets of Views

As noted above, while the join of two views need not exist, we at least have a construction for the join when it does exist. The situation for meets is even less pleasant. Not only do they not always exist, but there does not appear to be any reasonable construction for them even when they do exist.

For similar reasoning as in join construction, we cannot simply look for the lub of two views in the natural ordering to obtain a meet. Rather, we must start with a definition specifying that which we want from meet in a decomposition framework. We therefore ask that any meet Γ of the views Γ_1 and Γ_2 have the following properties.

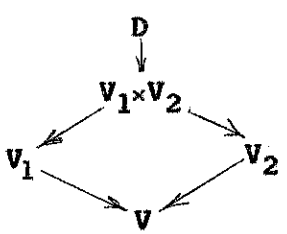
(i) It must be less than both Γ_1 and Γ_2; *i.e., both* Γ_1 and Γ_2 *individually* must completely determine it.

(ii) It must record any interaction of Γ_1 and Γ_2.

The first property says that it is small enough; it is guaranteed if we respect the view ordering. The second property says that it is not too small; our task is to formalize this notion.

3.3.1 DEFINITION Let $\Gamma = (\mathbf{V},\gamma)$ be a view of $\mathbf{D}$ such that $\Gamma \leq \Gamma_1$ and $\Gamma \leq \Gamma_2$.

(a) The Γ-*product* of Γ_1 and Γ_2, denoted $\Gamma_1 \otimes_\Gamma \Gamma_2$, is the pair $(\mathbf{V}_1 \otimes_\Gamma \mathbf{V}_2, \gamma_1 \times \gamma_2)$, where $\mathbf{V}_1 \otimes_\Gamma \mathbf{V}_2$ has the same relation symbols as the free product, but has as constraints the axioms $\mathbf{C}_1 \cup \mathbf{C}_2$, together with the constraints induced by the requirement that the two paths from $\mathbf{V}_1 \times \mathbf{V}_2$ to $\mathbf{V}$ in the diagram below be the same.



More precisely, for each relation symbol R of **V**, recall that $\psi_R^\gamma 1$ (resp. $\psi_R^\gamma 2$) denotes the interpretation formula of R in $\mathbf{V}_1$ (resp. $\mathbf{V}_2$). The induced axiom for R is $\psi_R^\gamma 1 \iff \psi_R^\gamma 2$. We have one such axiom for each R.

Intuitively, the Γ-product of Γ_1 and Γ_2 takes the free product $\Gamma_1 \times \Gamma_2$ and adds to it exactly the interdependence axioms which are required to make each component determine the same instance of **V**. If the resulting view is surjective, this means that modulo a given state of the view Γ, Γ_1 and Γ_2 are independent. We can thus use this to define meet of views.

3.3.2 DEFINITION Let $\Gamma = (\mathbf{V},\gamma)$ be a view of **D**. Γ is a *meet* of Γ_1 and Γ_2 if it satisfies the following two properties.

(i) $\Gamma \leq \Gamma_1$ and $\Gamma \leq \Gamma_2$.

(ii) The Γ-product of Γ_1 and Γ_2 is a view of **D**; *i.e.*, it is surjective.

3.3.3 THEOREM (a) *The meet of* Γ_1 *and* Γ_2, *if it exists, is unique up to isomorphism. In this case, we denote it by* $\Gamma_1 \wedge \Gamma_2$. □

3.3.4 EXAMPLE Consider again the Example of 3.2.4. Intuitively, the meet Γ of Γ_1 and Γ_2 should be the projection of the common column B; *i.e.*, $\Gamma = (\{R_2\},\phi),\pi_2)$. It is easily seen that this is indeed the case; the Γ-product induces the necessary inclusion dependencies $R_{12}[B] \subseteq R_{23}[B]$ and $R_{23}[B] \subseteq R_{12}[B]$.

3.3.5 EXAMPLE Continue with the example of horizontal decomposition of 3.2.7. It is easy to see that **0** is the meet of these two views. Thus, as we shall define below, these two views are independent.

Meets need not always exist. We illustrate with the following example.

3.3.6 EXAMPLE Let **D** be the schema with two unary relational symbols R and S, which take values over the same domain A. The only non-typing constraint is the inclusion dependency $R[A] \subseteq S[A]$. Let Γ_1 be the view π_R which throws away S but leaves R intact. Let Γ_2 similarly be the view π_S which throws away R, but leaves S intact. From a decomposition point of view, π_R and π_S are not independent, because any legal instance of R must be a subset of the current instance of S. However, there is no view Γ such that the Γ-product of these two views is surjective. In fact, it is not difficult to see that the only view which is less than both Γ_1 and Γ_2 is the zero view **0**. Hence Γ_1 and Γ_2 cannot have a meet.

3.4 Independence of Views

Our use of meet is primarily to characterize independence of views; we thus formalize this crucial concept.

3.4.1 DEFINITION Γ_1 and Γ_2 are *independent* if their meet exists and is **0**.

Note that, for $\Gamma = \mathbf{0}$, the Γ-product of Γ_1 and Γ_2 reduces to the free product. This is in harmony with our intuition; the meet is zero if and only if the two views can be updated completely independently, without any interrelational constraints. We note also that this definition coincides with that of Bancilhon and Spyratos [BaSp81], who define two views to be independent if the product of their images is the image of their product.

Example 3.3.4 may seem at odds with traditional decomposition theory. Γ_1 and Γ_2 are the components in the most basic of decompositions; the projections of a relation which satisfies a join dependency. However, the components are not independent; the inclusion dependencies are present. What is true is that if we use null values appropriately, then our notion of independence makes sense. We illustrate with another example.

3.4.2 EXAMPLE Consider once again the schema of Example 2.2.3, modified to handle null values in the fashion outlined there. Define $\Gamma_1 = (\{S\},\{S[AB]\},\pi^\circ_{12})$ and $\Gamma_2 = (\{T\},\{T[BC]\},\pi^\circ_{23})$, where π°_{12} is the restrict-project view mapping which projects down the first two columns of just those tuples which do not have a null in column 1. π°_{23} is defined similarly. These two views are independent, since any legal state of Γ_1 can be combined with a legal state of Γ_2 to give the image of a state of the original schema. Entries in one of the relations which do not have a match in the B column of the other relation have a entry with a null in the projected column.

In [BaSp81], Bancilhon and Spyratos define two views to be *weakly independent* if the set of permissible updates to one are independent of the state of the other. Translated to our framework, Γ_1 and Γ_2 are weakly independent if they have a meet. For then, all we need do to ensure that an update will not cause a problem in the other component is to make sure that the image of the new state under the (unique) view morphism $f : \Gamma_1 \rightarrow \Gamma_1 \wedge \Gamma_2$ is the same as the image of the old state. Under this definition, the two views of Example 3.3.4. are weakly independent, but those of Example 3.3.6 are not. However, note that any view is weakly independent of itself, so this concept of "independence" must be used with discretion.

3.5 Complements of Views

In decomposing a schema **D** into two components Γ_1 and Γ_2, we want the join of the components to determine the entire schema, and we want the components to be independent. In other words, we want their join to be **1** and their meet to be **0**. We thus generalize the lattice theoretic notion of complement to reflect this.

3.5.1 DEFINITION Γ_2 is a *complement* of Γ_1 if both the meet and the join of these two elements exist, and if

(i) $\Gamma_1 \vee \Gamma_2 = \mathbf{1}$

(ii) $\Gamma_1 \wedge \Gamma_2 = \mathbf{0}$.

3.5.2 EXAMPLE Let us reexamine Example 3.4.2. The two views Γ_1 and Γ_2 are independent. However, they do not quite determine the original schema. This is easy to illustrate by showing how the mappings would treat single tuples. If the single tuple (a,b) is the state of Γ_1 and (b,c) is the state of Γ_2, there are four possibilities for the resulting state of the base schema:

(i) (a,b,c)

(i) (a,b,η) and (a,b,c);

(ii) (η,b,c) and (a,b,c);

(iv) (a,b,η), (η,b,c) and (a,b,c).

To avoid this ambiguity, we would need to add a further axiom to the base schema eliminating all but one of the possibilities. Following Sciore [Scio80] or Zaniolo [Zani82], we may say that tuple t_1 *subsumes* t_2 if for every position in t_2 which is not null, the corresponding position in t_1 is

not null either, and has the same value. We need to axiomatize the degree of admissibility of subsumed tuples in the base view. (i) allows no subsumed tuples, (iv) requires all subsumed tuples, while (ii) and (iii) are intermediate. Once we define such admissibility by a formal constraint on the schema, we have a decomposition. Of course, all four of these schemata are isomorphic to each other, but the original schemata is not isomorphic to any of them.

Unfortunately, the situation is not always so pleasant. Complements need not always exist, nor need they be unique when they do. We illustrate with examples.

3.5.3 EXAMPLE Consider the schemata of Example 3.3.4. It is not difficult to see that there is no complement to Γ_1, the projection of **R** onto its first two columns. Any complement would have to carry enough information to reconstruct the relation; it is easy to see that Γ_2, the projection onto the last two columns, is minimal in that regard. But these two views are not independent, so there can be no complement.

3.5.4 EXAMPLE Let **D** be the schema with three unary relations R[A], S[A], and T[A]. The only constraint on **D** is that an element either appears in none of the relations or in exactly two of them. Let Γ_R be the view which forgets S and T, but leaves R intact. Define Γ_S and Γ_T similarly. Then any two of these views are complementary, so that complements need not be unique. In this situation, it is questionable whether, say, Γ_R and Γ_S should be regarded as a decomposition of **D**. There is the third view Γ_T which is independent of either one individually, but which is completely determined by both together.

4. THE DECOMPOSITION RESULTS

The results of the previous section show that we cannot hope to develop a general algebraic theory of database schema decomposition covering all views of all schemata. To have a usable decomposition theory, we must restrict our attention to a more tractable class of schemata and views. In this section, we develop decomposition results for one such class.

Throughout this section, we fix a database framework $T = (\mathbf{T},\mathbf{K},\mathbf{A})$.

4.1 Order-Theoretic Properties of Database Schemata

4.1.1 DEFINITION Let $\mathbf{D} = (\mathbf{R},\mathbf{C})$ be a database schema and let $M, N \in LDB(\mathbf{D},\mu)$. We define the analogs of the usual set operations in a componentwise fashion; *i.e.*, relation by relation.

(a) The *intersection* of M and N, denoted $M \cap N$, is defined for each $R \in \mathbf{R}$ by $R^{M\cap N} = R^M \cap R^N$. The intersection $\bigcap \Psi$ of an arbitrary nonempty family $\Psi \subseteq LDB(\mathbf{D},\mu)$ is defined analogously.

(b) The union of M and N, denoted $M \cup N$, is defined for each $R \in \mathbf{R}$ by $R^{M\cup N} = R^M \cup R^N$.

(c) Containment is defined componentwise also; $M \subseteq N$ if and only if $R^M \subseteq R^N$ for all $R \in \mathbf{R}$.

4.1.2 DEFINITION A database schema $\mathbf{D}$ has the *model intersection property* if and only if for any *nonempty* $\Psi \subseteq LDB(\mathbf{D},\mu)$, $\bigcap \Psi \in LDB(\mathbf{D},\mu)$ also.

We now show that a large class of the most commonly considered constraints yield schemata which satisfy the model intersection property.

4.1.3 DEFINITION Let $\mathbf{D} = (\mathbf{R},\mathbf{C})$ be a database schema.

(a) A constraint $C \in \mathbf{C}$ is *universal* if it is of the form $(\forall x_1)(\forall x_2)\ldots(\forall x_n)\Theta$, where Θ is a quantifier-free formula. C is *universal Horn* if, in addition, Θ is of the form $\phi_1 \vee \phi_2 \vee \ldots \vee \phi_k$, where each ϕ_i is either of the form $\neg R(x_1,x_2,\ldots,x_m)$ or $R(x_1,x_2,\ldots,x_m)$ for some $R \in \mathbf{RUT}$, and at most one of the ϕ_i's is of the latter form.

(b) $\mathbf{D}$ is *universal (resp. universal Horn)* if there is a set of universal sentences (resp. universal Horn sentences) Σ such that $\Sigma^+ = \mathbf{C}^+$. Note that the typing axioms are *not* included in this condition.

4.1.4 PROPOSITION *Let* $\mathbf{D}$ *be a database schema.* $\mathbf{D}$ *has the model intersection property if and only if it is a universal Horn schema.* □

Most of the constraints considered in database theory, including functional dependencies, join dependencies, and full template dependencies [SaU182] are universal Horn. The papers [Fagi82] and [GrJa82] contain extensive discussions of the properties of universal Horn constraints. The structure of a semilattice provides the appropriate algebraic foundations for models of schemata which satisfy the model intersection property.

4.1.5 SEMILATTICES A *(meet) semilattice* is an algebraic structure which has meets, much like a lattice, but which need not have joins. More precisely, a *meet semilattice* is a pair $\mathbf{S} = (S,\wedge)$, where $\mathbf{S}$ is a set and $\wedge$ is an associative, binary, idempotent operation on $\mathbf{S}$ [Grät78]. With $\mathbf{S}$ we may naturally associate a partial order $\leq$ defined by $s_1 \leq s_2$ if and only if $s_1 \wedge s_2 = s_1$. In other words, a meet semilattice corresponds to a partial order which has

glb's, but not necessarily lub's. We call a meet semilattice **S** *almost complete* if every *nonempty* subset has a glb. (**S** would be *complete* if it were to have glb's of all subsets, including the empty set. If this were the case, it would necessarily be a lattice [Grät78, I.3, Lem. 14].)

4.1.6 PROPOSITION *Let* ***D*** *be a database schema which has the model intersection property. Then* LDB(**D**) *forms an almost complete meet semilattice under* $\cap$, *with* $\subseteq$ *defining the induced ordering. The least element is the intersection of all models, which we denote by* $\perp$. □

In most cases, $\perp$ will be the empty model ϕ. However, sentences of the form $(\forall x)R(x)$ are universal Horn, and hence admissible within our framework. They would seem to have little place within database theory, and could be excluded, if desired.

LDB(**D**,μ) not in general a lattice. For example, let **D** be the schema with the single two-column relation R[AB]. The only constraint is the functional dependency $A \rightarrow B$. The single tuple relations {(x,y)} and {(x,z)} are clearly in LDB(**D**,μ), yet there is no legal database which contains both tuples. Hence these two databases cannot have an upper bound. However, there is a "join" result of sorts. Any set of legal databases which are known to be contained in a common legal database does have a lub: the intersection (meet) of all the legal databases which contain the union.

4.1.7 DEFINITION Let **D** be a database schema and let $M \in DB(\mathbf{D},\mu)$. A *completion* of M is an $N \in LDB(\mathbf{D},\mu)$ such that:

(i) $M \subseteq N$.

(ii) If $P \in LDB(\mathbf{D},\mu)$ with $M \subseteq P \subseteq N$, then $P = N$.

4.1.8 PROPOSITION *Let* **D** *be a database schema with the model intersection property, and let* $M \in DB(\mathbf{D},\mu)$. *Then* M *has a unique completion, denoted* $\hat{M}$, *if and only if there is an* $N \in LDB(\mathbf{D},\mu)$ *such that* $M \subseteq N$. *In this case the completion is given by* $\cap\{P \in LDB(\mathbf{D},\mu) \mid M \subseteq P\}$. □

The idea of completion of a database schema is not new. The above definition and proposition are essentially the same as those given in Maier, *et al* [MaMS79], who considered the concept in the framework of constraint inference. Sciore [Scio80] has also investigated the concept of completion in his more general framework of objects.

The concept of a continuous function on a complete lattice has played a major role in the development of the theory of programming language semantics in recent years. A similar notion of a continuous function on an almost complete semilattice plays a key role in the development of our decomposition results.

4.1.9 DEFINITION Let $\mathbf{S} = (S,\wedge)$ and $\mathbf{S}' = (S',\wedge)$ be almost complete meet semilattices. $[\mathbf{S} \to \mathbf{S}']$ denotes the set of all functions from S into S'. For $f, g \in [\mathbf{S} \to \mathbf{S}']$, define $f \leq g$ if and only if $f(s) \leq g(s)$ for all $s \in S$. This ordering makes $[\mathbf{S} \to \mathbf{S}']$ an almost complete meet semilattice; for F a nonempty subset of $[\mathbf{S} \to \mathbf{S}']$, define $(\wedge F)(s)$ to be the function $s \mapsto \bigwedge_{f \in F} f(s)$.

A subset of S is *directed* if it contains the meet of any two of its elements. A function $f : S \to S'$ is *continuous* if it preserves (arbitrary, not just finite) meets of directed sets. More precisely, f is continuous if

for any *nonempty* directed set D, we have that $f(\bigwedge_{d \in D} d) = \bigwedge_{d \in D} (f(d))$.

4.1.10 DEFINITION Let **D** be a database schema with the model intersection property, and let $\Gamma = (\mathbf{V}, \gamma)$ be a view of **D**. Γ is *continuous* precisely when:
(i) **V** also has the model intersection property, and
(ii) γ' is a continuous function from LDB(**D**,μ) into LDB(**V**,μ).

The continuity property says that the database mapping respects model intersections. This means first of all that the mapping is monotonic: smaller databases in the base schema map to smaller databases in the view. More importantly, however, it implies that that given any legal database N of the view, there must be a least (with respect to the lattice ordering) legal database in the base schema which maps to N.

4.1.11 EXAMPLE Reconsider the schema and views of Example 3.2.4. The single relation schema R[ABC] constrained by the join dependency ⋈[AB,BC]. Since a join dependency is universal Horn, the schemata all satisfy the model intersection property. However, neither of the projections π_{12} or π_{23} is continuous. If the state of the view Γ_1 is {(a,b)}, then the state of the base schema must be a set of tuples of the form (a,b,?). However, there is no such tuple which is in all such states, so the intersection of all states of the base which map to {(a,b)} is Φ. Hence the view map is not continuous. Similarly, Γ_2 is not continuous.

4.1.12 EXAMPLE Consider again the example begun in 2.2.3, and continued in 3.4.2 and 3.5.2. This is the previous example, augmented to use null values.

All of the variations are easily seen to be universal Horn, and hence all have the model intersection property. However, only configuration (iv) of 3.7.3 gives the two views Γ_1 and Γ_2 the property of continuity. For example, suppose that the tuple (a,b) is in the state of the view Γ_1. We know that there is some tuple (a,b,?) in the base schema state. To satisfy the continuity, there must be such a tuple *in all* legal states which have (a,b) in their image. The only way we can ensure this is to require that (a,b,η) be in the state of the base regardless of whether or not there is another tuple of the form (a,b,c). Hence, to make the idea of continuity work with project-join decompositions, we must include all tuples which can carry partial information for the view.

4.1.13 EXAMPLE The above idea can be extended to any join dependency [BeVa81] [Scio82]. For the sake of concreteness, consider the schema with the single 4-ary relation symbol R[ABCD] and the join dependency ⋈[AB,BC,CD]. Augment this schema to allow null tuples of the form (a,b,η,η), (η,b,c,η), and (η,η,c,d), and require that the tuple (a,b,c,d) be present if and only if each of the above three is. The resulting schema is universal Horn, and any of the three projections π°_{12}, π°_{23}, and π°_{34} onto non-null tuples only is continuous. In this case, since the join is a cascade, we can also allow tuples of the form (a,b,c,η) and (η,b,c,d) if we wish. (We must either mandate their inclusion or exclusion, however.) Thus, for a cascaded join dependency, there is more than one possible augmentation with nulls.

4.1.14 EXAMPLE A restriction is always continuous. For example, each of the views in Example 3.2.7 is continuous.

Let **D** be a database schema having the model intersection property, and let $\Gamma = \{ \Gamma_1 = (\mathbf{V}_1,\gamma_1), \Gamma_2 = (\mathbf{V}_2,\gamma_2)\}$ be a decomposition of **D** into continuous independent components. It is not difficult to see that the Γ_i's are also have the model intersection property. We have an isomorphism $f : \mathbf{V}_1 \times \mathbf{V}_2 \rightarrow \mathbf{D}$, since $\mathbf{V}_1$ and $\mathbf{V}_2$ are independent. Now add the further axiom to $\mathbf{V}_1 \times \mathbf{V}_2$ that the $\mathbf{V}_2$ part must be $\perp$. Since Γ_2 is universal, and since the two views are independent, $\mathbf{V}_1$ can still assume any of its legal states. Therefore, we essentially have an immersion $f_1 : \mathbf{V}_1 \rightarrow \mathbf{D}$ defined by $f : \mathbf{V}_1 \times \perp \rightarrow \mathbf{D}$. It is clear that $f \circ f_1$ is the identity mapping on $\mathbf{V}_1$. This immersion is in a sense "least" in that $\mathbf{V}_2$ was set to $\perp$ to construct it; it is "independent" of $\mathbf{V}_2$. The formalization of this property plays a key role the development of our decomposition theory.

4.1.15 DEFINITION A left inverse is called a retraction. More precisely, let **S** and **S'** be almost complete semilattices, as above, and let $f \in [\mathbf{S} \rightarrow \mathbf{S}']$. f is a *retraction* if there is a $g \in [\mathbf{S}' \rightarrow \mathbf{S}]$ such that $f \circ g = 1_{\mathbf{S}'}$. Now let Θ be a subset of $[\mathbf{S}' \rightarrow \mathbf{S}]$, and let $g \in \Theta$. g is *least* with respect to Θ if for any $h \in \Theta$ such that $f \circ h = 1_{\mathbf{S}'}$, we have that $g \leq h$. Such a least retraction is clearly unique, if it exists.

4.1.16 DEFINITION Let **D** be a database schema, and let $\Gamma = (\mathbf{V},\gamma)$ be a view of **D**. Assume that both **D** and Γ have the model intersection property. Γ is *least retractable if* γ' is a retraction which has a least inverse with respect to the subset of $[\mathrm{LDB}(\mathbf{V},\mu) \rightarrow \mathrm{LDB}(\mathbf{D},\mu)]$ consisting of mappings which are database morphisms. In the case that Γ is least retractable, we denote the least inverse to γ by $\gamma^{\#}$. The composition $\gamma^{\#} \circ \gamma$ is called the *endomorphism of* Γ, and is denoted by $\gamma^{@}$.

Continuing with the example preceding 4.1.15, note that the on $f_1(\mathbf{V_1}) = \gamma_1^{\#}(\mathbf{V_1})$, the view mapping $\gamma_1 : \mathbf{D} \rightarrow \mathbf{V_1}$ looks just like an identity mapping. In other words, the part of **D** that is the image of the least retraction is an exact copy of the view itself. Thus, the composition $\gamma^{\#} \circ \gamma = \gamma^{@}$ looks exactly like a projection of the view into the main schema. This is a key property of component views.

4.1.17 EXAMPLE Consider our continuing example of join decompostion, modified to be continuous as outlined in 4.1.12. The least inverse of the projection π_{12}° is just the embedding defined by $(a,b) \mapsto (a,b,\eta)$. The corresponding endomorphism $(\pi_{12}^{\circ})^{@}$ is the database mapping which saves tuples of the form (a,b,η), and discards all others.

4.1.18 EXAMPLE Consider the horizontal decomposition Example 3.2.7. The least inverse of the horizontal projection $\sigma_{2,nc}$ is just the identity embedding. The corresponding endomorphism strips away tuples (a.b) with $\tau_c(b)$ and leaves intact those with $\tau_{nc}(x)$.

4.1.19 DEFINITIONS Let $\mathbf{S} = (S,\wedge)$ be an almost complete semilattice, and let $f \in [\mathbf{S} \rightarrow \mathbf{S}]$. A *fixpoint* of f is any $s \in S$ such that $f(s) = s$. Fixpt(f) denotes the set of all fixpoints of f. We say that f is *downward stationary* if whenever $s \in \text{Fixpt}(f)$ and $s' \leq s$, we have that $s' \in \text{Fixpt}(f)$ also. f is *idempotent* if $f = f \circ f$.

4.1.20 DEFINITION Let **D** be a least retractable database schema, and let $\Gamma = (\mathbf{V},\gamma)$ be a view of **D**. Γ is *downward stationary* when $(\gamma^{@})'$ is.

4.1.21 EXAMPLES The views discussed in both Examples 4.1.17 and 4.1.18 are downward stationary and idempotent. For example, consider $(\pi^{\circ}_{12})^{@}$. A fixpoint of this mapping is any database consisting entirely of tuples of the form (a,b,n). Clearly, any subset of such a database is also of this form; hence the endomorphism is downward stationary. Since the image under $(\pi^{\circ}_{12})^{@}$ of any state is such a fixpoint, the view is also idempotent.

4.2 The Algebra of Strong Endomorphisms

The schemata which we will use in our decomposition framework are precisely those which satisfy all of the properties which we have just developed.

4.2.1 DEFINITION Let **D** be a database schema which satisfies the model intersection property.

(a) A view Q of **D** is called a *strong view* if it is continuous, least retractable, idempotent and downward stationary. SView(**D**) denotes the set of all strong views of **D**.

(b) A database morphism $f: \mathbf{D} \rightarrow \mathbf{D}$ is called a *strong endomorphism* if it is continuous, idempotent, and downward stationary. We let SEndo(**D**) denote the set of all strong endomorphisms of **D**.

For the rest of this section, we shall fix a database schema $\mathbf{D} = (\mathbf{R},\mathbf{C})$ *which satisfies the model intersection property.*

We wish to continue to identify isomorphic views. However, since we now need to use the order structure as well, we must make certain that isomorphic views have isomorphic order properties. Fortunately, this is the case.

4.2.2 LEMMA *Let* $\Gamma_1 = (\mathbf{V_2}, \gamma_1)$ *and* $\Gamma_2 = (\mathbf{V_2}, \gamma_2)$ *be strong views of* **D**, *and let* $f : \Gamma_1 \rightarrow \Gamma_2$ *be a view morphism. Then,* $f' : \mathrm{LDB}(\mathbf{V_1}, \mu) \rightarrow \mathrm{LDB}(\mathbf{V_2}, \mu)$ *is a continuous mapping of semilattices.* □

4.2.3 COROLLARY *If* Γ_1 *and* Γ_2 *are isomorphic, then the isomorphism* $i : \Gamma_1 \rightarrow \Gamma_2$ *respects the order structure. Thus, any strong view of* Γ_1 *may also be regarded as a strong view of* Γ_2. ■

4.2.4 DEFINITION QSView(**D**) denotes the set of equivalence classes of SView(**D**) under the view ordering relation $\preceq$ of 3.1.4.

Note that the equivalence classes of QSView(**D**) will, in general, be subclasses of those of QView(**D**), as a strong view may well be isomorphic to one which is not strong. What the above corollary does establish is that ordinary isomorphisms between strong views are strong isomorphisms, in that they preserve the ordering properties of the views. Thus, we do not need to develop a special notion of isomorphism for strong views.

4.2.5 MAIN REPRESENTATION THEOREM *There is a natural bijective correspondence between* QSView(**D**) *and* SEndo(**D**), *defined as follows.*

(a) *Given a strong view* $\Gamma = (\mathbf{V}, \gamma)$, *the associated strong endomorphism is just* $\gamma^{@}$.

(b) *Given a strong endomorphism f, let* $\mathbf{D}_f$ *denote the faithful closure of* **D** *under f. Denote the resulting faithful morphism by* $f^{\$} : \mathbf{D} \rightarrow \mathbf{D}_f$. *The pair* $\Xi_f = (\mathbf{D}_f, f^{\$})$ *is then a strong view of* **D**, *which is the image of f under the correspondence.*

(c) *The correspondence is natural in that for any* $f \in SEndo(\mathbf{D})$, $(f^{\$})^{@} = f$. □

Theorem 4.2.5 says that we can study strong views by studying strong endomorphisms. The latter are much easier to work with because they are morphisms with the same domain and codomain.

4.2.6 LEMMA *Let* $f, g \in SEndo(\mathbf{D})$.

(a) $f \circ g \in SEndo(\mathbf{D})$, $f \circ g = g \circ f$, *and for any* $M \in LDB(\mathbf{D},\mu)$, $(f \circ g)'(M) = f'(M) \cap g'(M)$.

(b) *There is a unique* $h \in SEndo(\mathbf{D})$ *such that for any* $M \in LDB(\mathbf{D},\mu)$, $h'(M) = (g'(M) \cup h'(M))\hat{}$. *This* h *is denoted by* f+g *and is called the* sum *of* f *and* g. □

4.2.7 MAIN STRUCTURE THEOREM *With meet defined as composition and join defined as sum,* SEndo(**D**) *forms a distributive lattice with a largest element (the identity view* $\mathbf{1}_{\mathbf{D}}$*) and a least element (the zero view* $\mathbf{0}_{\mathbf{D}}$*). Upon translating to* QSView(**D**) *using the natural isomorphism of the main representation theorem,* QSView(**D**) *becomes a distributive lattice of views in which meet and join are correct view meet and join, in the sense of* 3.2.2 *and* 3.3.2. *More precisely, for any* $f, g \in SEndo(\mathbf{D})$, $\mathfrak{S}_{f \circ g}$ *is the meet of* $\mathfrak{S}_f$ *and* $\mathfrak{S}_g$, and $\mathfrak{S}_{f+g}$ *is the join of* $\mathfrak{S}_f$ *and* $\mathfrak{S}_g$. □

This framework lends set-operation plausiblity to the notions of meet and join. The intuitive notion of meet is the intersection of the information of the two views; in this framework, it translates to the intersection of the models. Similarly, the intuitive notion of the join is the union of the information of the two components, together with anything inferable from this

union; in this framework, it translates to union of the models, with the closure operation accounting for the inferred information.

To use the main decomposition tool 1.3, we need to work with a lattice whose complemented elements are finite in number. It is not known to us whether or not SEndo(**D**) has this property. However, we can restrict attention to a large class of reasonable strong endomorphisms which are finite in number.

4.2.8 DEFINITION (a) Let $f \in$ SEndo(**D**). f is a *restriction* if, for each $R \in$ **R**, the defining formula $\psi_R^f(x_1,x_2,\ldots,x_n)$ is of the form

$$R(x_1,x_2,\ldots,x_n)\wedge\tau_1(x_1)\wedge\tau_2(x_2)\wedge\ldots\tau_n(x_n)$$

for some $\tau_1,\tau_2,\ldots,\tau_n \in$ **T**. f is an *algebraic endomorphism* if it is the finite join of restrictions. A strong view is algebraic if it corresponds to an algebraic endomorphism. The set of all algebraic endomorphisms of **D** is denoted Alg(**D**).

4.2.9 THEOREM Alg(**D**) *forms a finite distributive sublattice of* SEndo(**D**). *Thus, by 1.3, any schema* **D** *with the model intersection property has a unique maximal decomposition into algebraic views.* □

Our concept of restriction is much more extensive than the classical notion. For example, projections which are used in decomposition are really restrictions to null fields, so that we actually cover both horizontal and vertical decomposition. We illustrate with an example.

4.2.10 EXAMPLE Consider the schema and views of 4.1.13. Each of the "projections" π_{12}°, π_{23}°, and π_{34}° is in fact a restriction. For example, $(\pi_{12}^\circ)^@$

is the restriction requiring the entries of columns 1 and 2 to be non-null, and the entries of columns 3 and 4 to be null. These three projections are the atomic components of the ultimate decomposition. The composite views are not restrictions, however. The join of $(\pi^{\circ}_{12})^{@}$ and $(\pi^{\circ}_{23})^{@}$ yields those tuples which have a null in column 4, and in at most of columns 1 and 2, which cannot be expressed as a single restriction. If we allow tuples of the form (a,b,c,η), then this join looks just like the schema of Example 4.1.12. Note that we can also take the join of $(\pi^{\circ}_{12})^{@}$ and $(\pi^{\circ}_{34})^{@}$, even though these are not joinable in the classical sense. We get all tuples of the form (a,b,η,η) and (η,η,c,d).

4.2.11 EXAMPLE The true horizontal decomposition of 2.2.2 and 3.2.7 fits into this framework in the obvious way.

Given the host of properties that algebraic views have, one might conjecture that they are all complemented. The following example shows that this is not the case.

4.2.12 EXAMPLE Reconsider Example 3.3.6. The inclusion dependency R[A]⊆S[A] is clearly universal Horn, so **D** satisfies the model intersection property. The view π_2 is a strong view; $\pi_2^{@}$ is just the morphism which leaves S intact but leaves R empty. However, π_2 cannot have a complement. If it did, by the definition of join of strong endomorphisms, it would have to be π_1. However, these two views do not even have a meet. Note also that π_1 is not a strong view; it is not downward stationary.

5. CONCLUSIONS

We have developed a set of mathematical tools which allow us to describe the notion of relational database decomposition in algebraic terms. The nonexistence of joins and meets of arbitrary schemata suggest that a general theory of decompostion which applies to all schemata and which is sufficiently powerful to yield useful results is not achievable. Rather, attention must be directed to more specific classes of schemata. We developed the algebraic views as one such class. We showed that the usual notions of vertical decomposition, as well as horizontal decomposition, fit into this framework. However, this is only a start. The work leaves many open problems. We suggest a few.

1. Under suitable conditions, can algorithms be found to determine the ultimate components of a schema?
2. Although they are closely related, decomposition and normalization are not the same. What does our theory of decomposition have to say about normalization?
3. Can the results for decomposition of strong views be extended to a wider class of schemata?

One of the very positive things which this general formulation of decomposition has revealed is the exact nature of the notion of independence of views of a database schema. By taking a very general look at this concept, we have identified features of it that more specialized approaches have missed. Specifically, our general definition of meet has indicated that, at least in the classical case of decomposition into projections, independence is intimately related to the way in which null values are used. We give one final example to illustrate this.

5.1 EXAMPLE Consider the relational schema with the single relation symbol R[ABC] and the triad of functional dependencies { A → B, B → C, C → A }. In other words, every attribute functionally determines every other attribute. By the rules for inferring join dependencies from functional dependencies [BeVa81] [Scio82], each of the join dependencies ⋈[AB,AC], ⋈[AB,BC], ⋈[AC,BC], and ⋈[AB,AC,BC] is true. Therefore, we have, under the classical definitions, four ways to decompose this schema. All of the recent formulations of decomposition into independent components [BeRi80] [ArCa78] [MMSU81] [GrYa82] would exclude the last join dependency as redundant. However, they would all admit the other three. Since the problem is completely symmetric, there is apparently no way to choose. Our approach takes a different point of view. None of these decompositions are actually into independent components; the interrelation column inclusion dependencies are there. To get a decomposition into truly independent components, we must augment the schema with null values. What is interesting is that there is a unique augmentation for each of the three cases. For example, to get the decomposition into AB and AC, we allow tuples to have nulls in either column 1 or column 3, but not both. The functional dependencies hold on non-null elements. *Thus, to select the appropriate decomposition, the schema designer must decide exactly how null values are to be used.* Rather than a nuisance, we believe that such attention to null use will be beneficial. If the relation is to be decomposed into the components of AB and AC, then there *must* be an appropriate semantics for tuples of the form (η,b,c) and (a,b,η). If not, the decomposition does not make sense in the first place. Although other authors have investigated decomposition of relations with null values (*e.g.*, [Lien79]), the nulls were assumed to be in place from the start. Our claim is that, given a desired

decomposition based upon a join dependency, there are precise rules for specifying how nulls should be used, in order that the decomposition be into truly independent components.

REFERENCES

[AhBU79] Aho, A. V., C. Beeri, and J. D. Ullman, "The theory of joins in relational databases", *ACM TODS*, **4**,3(1979), pp. 297-314.

[ArCa78] Arora, A. K., and C. R. Carlson, "The information preservation properties of relational database transformations", *Proc. 1978 VLDB Conf.*, pp. 352-359.

[BaSp81] Bancilhon, F. and N. Spyratos, "Independent components of databases", *Proc. 1981 VLDB Conf.*, pp. 398-408.

[BeBG78] Beeri, C., P. A. Bernstein, and N. Goodman, "A sophisticate's introduction to database normalization theory", *Proc. 1978 VLDB Conf.*, pp.113-124.

[BeRi80] Beeri, C, and J. Rissanen, *Faithful Representation of Relational Database Schemes*, IBM Research Report RJ2722, 1980.

[BeVa81] Beeri, C., and M. Vardi, "On the properties of join dependencies", in *Advances in Data Base Theory, Vol. 1*, ed. by H. Gallaire, J. Minker, and J. M. Nicolas, Plenum Press, 1981, pp. 25-71.

[Birk67] Birkhoff, G., *Lattice Theory, Third Edition*, American Mathematical Society, 1967.

[CaFP82] Casanova, M. A., R. Fagin, and C. H. Papadimitriou, "Inclusion dependencies and their interaction with functional dependencies", *Proc. Principles of Database Systems Conf. (1982)*, pp. 171-176.

[Ende72] Enderton, H. B., *A Mathematical Introduction to Logic*, Academic Press, 1972.

[Fagi82] Fagin, R., "Horn clauses and database dependencies", *JACM*, **29**,4(1982), pp. 952-985.

[Grät78] Grätzer, G., *General Lattice Theory*, Academic Press, 1978.

[GrYa82] Graham, M. H., and M. Yannakakis, "Independent database schemas", *Proc. Principles of Database Systems Conf.* (1982), pp. 199-204.

[Gran82] Grant, J., *Adequate Database Transformations*, TR-1145, Towson State University, February 1982.

[GrJa82] Grant, J., and B. E. Jacobs, "On the family of generalized dependency constraints", *JACM*, **29**,5(1982), pp. 986-997.

[Halm74] Halmos, P. R., *Lectures on Boolean Algebras*, Springer-Verlag, 1974.

[Jaco79] Jacobs, B. E., *Applications of Database Logic to Conceptual and External View Protection*, U. of Md. Comp. Sci. Tech. Report 815, Sept. 1979.

[Jaco82] Jacobs, B. E., "On database logic", *JACM*, **29**,2(1982), pp. 310-332.

[JaAK82] Jacobs. B. E., A. R. Aronson, and A. C. Klug, "On interpretations of relational languages and solutions to the implied constraint problem", *ACM TODS*, **7**,2(1982), pp. 291-315.

[Kent81] Kent, W., "Consequences of assuming a universal relation", *ACM TODS*,**6**,4(1981), pp. 539-556.

[Lien79] Lien, Y. E., "Multivalued dependencies with null values in relational databases", *Proc. 1979 VLDB Conf.*, pp. 61-66.

[McMi79] McSkimin J. R. and J. Minker, "A predicate calculus based semantic network for deductive searching", in *Associative Netowrks*, ed. by N. V. Findler, Academic Press, 1979, pp. 205-238.

[MaMS79] Maier, D., A. O. Mendelzon, and Y. Sagiv, "Testing implications of data dependencies", *ACM TODS*, **4**,4(1979), pp. 455-469.

[MMSU81] Maier, D., A. O. Mendelzon, F. Sadri, and J. D. Ullman, "Adequacy of decompositions of relational databases", in *Advances in Data Base Theory, Vol. 1*, ed. by H. Gallaire, J. Minker, and J. M. Nicolas, Plenum Press, 1981, pp. 101-114.

[Monk76] Monk, J. D., *Mathematical Logic*, Springer-Verlag, 1976.

[Reit80] Reiter, R., "Equality and domain closure for first-order databases", *JACM*, **27**,2(1980), pp. 235-249.

[Riss77] Rissanen, J., "Independent components of relations", *ACM TODS*, **2**,4(1977), pp. 317-325.

[SaUl82] Sadri, F. and J. D. Ullman, "Template dependencies: a large class of dependencies in relational databases", *JACM*, **29**,2(1982), pp. 363-372.

[Scio80] Sciore, E., *The Universal Instance and Database Design*, TR #271, Princeton University, 1980.

[Scio82] Sciore, E., "A complete axiomatization of full join dependencies", *JACM*, **29**,2(1982), pp. 373-393.

[SmSm77] Smith, J. M., and D. C. P. Smith, "Database abstractions: aggregation and generalization", *ACM TODS*, **2**,2(1977), pp. 105-133.

[Smit78] Smith, J. M., "A normal form for abstract syntax", *Proc. 1978 VLDB Conf.*, pp. 156-162.

[Ullm80] Ullman, J. D., *Principles of Database Systems*, Computer Science Press, 1980.

[Zani82] Zaniolo, C., "Database relations with null values" *Proc. ACM Symposium on Principles of Database Systems*, 1982, pp. 27-33.