Using Powerdomains to Generalize Relational Databases*

Peter Buneman
Achim Jung
Atsushi Ohori†

Abstract

Much of relational algebra and the underlying principles of relational database design have a simple representation in the theory of domains that is traditionally used in the denotational semantics of programming languages. By investigating the possible orderings on powerdomains that are well-known in the study of nondeterminism and concurrency it is possible to show that many of the ideas in relational databases apply to structures that are much more general than relations. This also suggests a method of representing database objects as typed objects in programming languages.

In this paper we show how operations such as *natural join* and *projection* – which are fundamental to relational database design – can be generalized, and we use this generalized framework to give characterizations of several relational database concepts including functional dependencies and universal relations. All of these have a simple-minded semantics in terms of the underlying domains, which can be thought of as domains of partial descriptions of "real-world" objects. We also discuss the applicability of relational database theory to non-relational structures such as records with variants, higher-order relations, recursive structures and other ordered spaces.

1 Introduction

There are two motivations for this study. The first is to draw together a number of approaches to data models and to examine the extent to which they can be viewed

*This research was supported in part by grants NSF IRI86-10617, ONR NOOO-14-88-K-0634, ARO DAA6-29-84-k-0061. Atsushi Ohori was also supported in part by OKI Electric Industry Co., Japan.

†Authors addresses: Peter Buneman and Atsushi Ohori, Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd Street, Philadelphia, PA 19104-6389, USA; Achim Jung, Fachbereich Mathematik, Technische Hochschule Darmstadt, D-6100 Darmstadt, FRG

as generalizations of the relational data model. The second is to try to draw out the connection between data models and data types, something that is crucial if we are to achieve a proper integration of databases [4, 5, 39] and programming languages.

The main focus of this paper is the first of these. There are a number of attempts to generalize the relational data model beyond first-normal-form relations [17, 36, 32]; there are also numerous formulations of other data models [1, 18, 7, 19] that at first sight appear to have little to do with relations. We shall see that by exploiting the basic ideas of domain theory, well known in the study of semantics of programming languages, we can obtain generalizations of many of the basic results of relational databases in a way that has very little to do with the details of the data structures that are used to define them; and which allows the application of relational database principles to a much wider range of data models. Although some observations have been made [34, 14] that suggest a connection between database and programming language semantics, there appears to have been no attempt directly to characterize relational databases in the appropriate semantic domains.

To the hardened first-normal-form relational database theorist this paper offers little more than alternative, and perhaps simpler, derivations of some existing results. However, given the recent activity in the study of "higher order" relations, which attempts to apply the basic results of relational databases to other structures, it is interesting to ask how far this work can be pushed. What are the properties of the data model that allow us to define relational operators, functional dependencies etc.? In doing this, we shall find it useful to produce a simple denotational semantics for relations and other structures, which is an extension to the semantics for missing values proposed by Lipski [23]. The idea is that these structures denote sets of values in some space which we may think of as the "real world". One of the advantages of our approach is that it allows us to provide a denotational semantics for structures such as sets of attribute names, which usually receive an operational treatment. Such a semantics will, we hope, ultimately be useful if we are ever to achieve our second goal of achieving a healthy marriage of databases and programming languages.

The organization of this paper is as follows. In section 2 we describe the properties of the underlying domains that we shall need. Section 3 then shows how powerdomain orderings (orderings on sets of values) can be used to characterize the various joins that are discussed in relational algebra. In section 4, in trying to characterize projection, we introduce the notion of *schemes*, which generalize relational schemes (sets of column names). Schemes enjoy some nice properties with respect to power-domain orderings and allow us to characterize functional dependencies and universal relations, which is done in the following sections. Section 7 concludes by showing how these ideas can be applied to various extensions of relational databases including typed relations, relations with null values and various forms of higher order relations; it also suggests that there may be some limitations to what one can do with non first-normal-form relations. The reader who is more interested in data types and structures rather than some of the more esoteric areas of database theory may wish to skip much of sections 5 and 6, and turn directly to section 7.

2 Orderings and Domains

The idea that is fundamental in denotational semantics is that expressions denote values, and that the domain of values is partially ordered. In the same way we can think of database structures as descriptions and that these descriptions are partially ordered by how well they describe the real world. Without putting any particular structure on the real world, we can define the meaning $[\![d]\!]$ of a description d as the set of all real-world objects described by d. We can then say that a description d_1 is better than d_2, $d_1 \sqsupseteq d_2$, if d_1 describes a subset of the real-world objects described by d_2, i.e. $[\![d_1]\!] \subseteq [\![d_2]\!]$.

An example of such an ordering is to be found in flat record structures. A flat record is a partial function from a set $\mathcal{L}$ of labels to an unordered set $\mathcal{V}$ of values. If r_1 and r_2 are two such functions, then $r_1 \sqsupseteq r_2$ if the graph of r_1 contains the graph of r_2. For example,

$$\{Name \Rightarrow 'J.\ Doe'; Dept \Rightarrow 'Sales'; Office \Rightarrow 33\}$$
$$\sqsupseteq \{Name \Rightarrow 'J.\ Doe'; Dept \Rightarrow 'Sales'\}$$

Using the term “real world” to describe the semantics of such records is, of course, contentious. It is better to think of these records as partial descriptions (or approximations) to elements in some space or "universe" of total descriptions, in this case large – possibly infinite – record structures. Suppose that this universe were the function space $\mathcal{L} \rightarrow \mathcal{V}$ where $\mathcal{L} = \{Name, Dept, Office\}$, we would then have

$$[\![\{Name{\Rightarrow}'J.\ Doe';Dept{\Rightarrow}'Sales'\}]\!] =$$
$$\{\{Name{\Rightarrow}'J.\ Doe';Dept{\Rightarrow}'Sales';Office{\Rightarrow}\ v\} \ \mid v \in \mathcal{V}\}$$

Note that this formulation of the denotation of a record with incomplete information corresponds with that given in [23], and as it will shortly appear, this space of flat records provides the basis for the relational model; however there are a number of other orderings that we shall examine later in this paper. These include Bancilhon's complex objects [7], orderings on tree structures that give rise to higher order relations [17, 1, 36, 35, 32], the feature structures in unification-based grammar formalisms (see [43] for a survey), finite state automata [37], ψ-terms [2]. In this catalog we should also include Scott's aptly-named "information systems" – consistent, deductively closed sets of predicates [42]. In all of these it is possible to describe certain generalizations of relational operations.

We shall require somewhat more structure on our space $\mathcal{D}$ of partial descriptions than being partially ordered. The most important property is that it is *bounded complete*:

1. any non-empty subset S of $\mathcal{D}$ has a greatest lower bound $\sqcap S$,

In addition we shall also make two further assumptions that are common in denotational semantics [38]:

2. any directed subset S of $\mathcal{D}$ has a least upper bound $\sqcup S$,

3. the set $K(D)$ of *compact* elements in $\mathcal{D}$ forms a countable basis for $\mathcal{D}$.

Partially ordered sets $(\mathcal{D}, \sqsubseteq)$ with these properties are widely used in the semantics of programming languages, and are often called Scott domains [42]. Throughout this paper we shall refer to them as *domains*. We shall also use the notation $s_1 \sqcup s_2$ and $s_1 \sqcap s_2$ for $\sqcup\{s_1, s_2\}$ and $\sqcap\{s_1, s_2\}$ respectively.

It is an immediate consequence of the first condition that any subset of S of $\mathcal{D}$ that is bounded above has a least upper bound $\sqcup S$ and also that that $\mathcal{D}$ has a bottom element, $\bot_{\mathcal{D}}$. The second condition, when taken with the axiom of choice, ensures that every member of $\mathcal{D}$ is bounded above by some member of $\mathcal{D}_{max}$, the set of maximal elements of $\mathcal{D}$. We shall therefore use $\mathcal{D}_{max}$ as the universe of complete descriptions; and the definition of $[\![d]\!]$ is then simply $\{x \in \mathcal{D}_{max} \mid D \sqsubseteq x\} = \uparrow d \cap \mathcal{D}_{max}$. Apart from some remarks at the end of the paper, we shall not make any use of the third condition; however we should note that in any practical database context this condition will surely be satisfied.

There is one extra condition which we shall need when we introduce *schemes* below:

4. A domain $\mathcal{D}$ is *distributive* if every principal ideal $\downarrow x$ is a distributive lattice.

Note that the space of flat record structures is a distributive domain. Even more is true of this domain: each principal ideal is a complete atomic boolean algebra, that is, a powerset. We shall not need to assume this in general, however.

We shall see that there are a number of ways to construct domains that represent the kinds of data structures we use in databases; particularly simple are the *flat* domains. Given a set of atomic values $\mathcal{V}$, a flat domain $\mathcal{V}_\bot$ of $\mathcal{V}$ is obtained by adding bottom element $\bot$ to $\mathcal{V}$ and ordering them as $x \sqsubseteq y$ if and only if $x = y$ or $x = \bot$. This domain is a domain of atomic descriptions; an element $v \in \mathcal{V}_\bot$ is either a complete description ($v \neq \bot$) with the meaning $\{v\}$ or the non-informative description $\bot$ with the meaning $\mathcal{V}$. The bottom element introduced in $\mathcal{V}_\bot$ can be interpreted as a *null value* representing "unknown values". There are number of other approaches to null values, some of them distinguish "inappropriate" and "unknown" values. Such an approach is entirely consistent with what we develop here and can be modeled by domains that are more complicated than $\mathcal{V}_\bot$. Later we shall comment more on null values.

We can now describe more precisely the domain of *labeled records* that we discussed in the introduction. Given a countable set of labels $\mathcal{L}$ and a domain $\mathcal{D}$, a domain of labeled record $\mathcal{L} \to \mathcal{D}$ over $\mathcal{D}$ is a set of total functions from $\mathcal{L}$ to $\mathcal{D}$ with the ordering defined as $r_1 \sqsubseteq r_2$ if and only if for all $l \in \mathcal{L}$, $r_1(l) \sqsubseteq r_2(l)$. This can be thought of as a domain of descriptions by attributes. This ordering represents the fact that r_2 is a better description that r_1 if r_2 has better descriptions than r_1 in all attributes. The minimal element $\bot_{\mathcal{L}\to\mathcal{D}}$ in $\mathcal{L} \to \mathcal{D}$ is the constant function $\bot_{\mathcal{D}}$ and if S is a set of functions, then $\sqcap S$ is the function r such that for all l, $r(l) = \sqcap\{s(l) \mid s \in S\}$, and $\sqcup S$ is the function r' such that for all $l, r'(l) = \sqcup\{s(l) \mid s \in S\}$ provided that all the least upper bounds exist.

The space of *flat records* is a special case of a domain of records where $\mathcal{D}$ is a flat domain $\mathcal{V}_\bot$. Indeed, the space of partial functions from $\mathcal{L}$ to $\mathcal{V}$ is isomorphic to $\mathcal{L} \to \mathcal{V}_\bot$. To make our notation for records precise, $\{l_1 \Rightarrow d_1; \cdots; l_n \Rightarrow d_n\}$ denotes an element r in $\mathcal{L} \to \mathcal{D}$ such that $r(l_i) = d_i$ for $1 \leq i \leq n$ otherwise $r(l) = \bot_{\mathcal{D}}$. For example, in $\mathcal{L} \to \mathcal{V}_\bot$, if

$$r_1 = \{\mathit{Emp\#}{\Rightarrow}12345;\mathit{Name}{\Rightarrow}\mathit{'J.\ Doe'}\}$$

and

$$r_2 = \{\mathit{Emp\#}{\Rightarrow}12345;\mathit{Sal}{\Rightarrow}20000\}$$

then

$$r_1 \sqcap r_2 = \{\mathit{Emp\#}{\Rightarrow}12345\}$$

and

$$r_1 \sqcup r_2 = \{\mathit{Emp\#}{\Rightarrow}12345;\mathit{Name}{\Rightarrow}\mathit{'J.\ Doe'};\mathit{Sal}{\Rightarrow}20000\}.$$

However $\{\mathit{Emp\#}{\Rightarrow}12345;\mathit{Name}{\Rightarrow}\mathit{'J.\ Doe'}\} \sqcup \{\mathit{Name}{\Rightarrow}\mathit{'K.\ Smith'}\}$ does not exist. An advantage of treating the space of flat records as $\mathcal{L} \to \mathcal{V}_\bot$ is that many results concerning flat records can be regarded as special cases of more general records and are readily applied to $\mathcal{L} \to \mathcal{D}$ for a more complicated domain $\mathcal{D}$.

As an example consider a database which lists the values of physical constants as they have been determined in particular experiments. Set up as a relational database, a typical entry might contain the following fields (among others): *author, publication, name of constant, lower bound, upper bound, dimension.* Being forced to express every record in first-normal-form has two obvious disadvantages. First, it does not

Name	Dept	Sal	Office
'K. Smith'	*'Mktg'*	30,000	275
'J. Doe'	*'Sales'*	20,000	147

$$\{\{Name \Rightarrow \text{'J. Doe'};\quad Dept \Rightarrow \text{'Sales'};\ Sal \Rightarrow 20{,}000;\ Office \Rightarrow 147\},$$
$$\{Name \Rightarrow \text{'K. Smith'}; Dept \Rightarrow \text{'Mktg'};\ Sal \Rightarrow 30{,}000\ \ Office \Rightarrow 275\}\}\ .$$

Figure 1: A relation and its representation as a set of records

reflect the property that the intervals [*lower bound, upper bound*] are partially ordered, smaller intervals being better approximations, and, second, there is no way how the obvious dependency (*name of constant* $\Rightarrow$ [*lower bound, upper bound*]) could be expressed in ordinary relational algebra. Our formalism as developed below will allow to state such a dependency and will provide a simple formula for checking the consistency of the database. (An inconsistency is reached in our example if asserted intervals for the same constant do not overlap.)

3 Powerdomains and Relational Algebra

Databases usually contain sets of values which, from our foregoing discussion, we would expect to describe sets of objects in the real world. If we interpret database values as elements in a domain, then database sets, such as relations, must be intepreted as sets of elements in that domain. Indeed, we can interpret a *first-normal-form* relation r of a relational scheme (a set of attribute names) R in the relational model as a set S of elements in the domain of flat records $\mathcal{L} \rightarrow \mathcal{V}_\perp$ such that for any $d \in S$, $\{l \mid d(l) \neq \perp\} = R$. Later in this section, we shall see that this interpretation is faithful to various relational operations and that the domain of flat records, therefore, serves as a domain of the relational model. This is how relations are described in languages such as Pascal/R [39], and extensions of this representation are to be found in Taxis [8] and Galileo [3]. Figure 1 shows a very simple relation and its representation as a set of flat records.

A	B
a	b
a	$\perp$

(i)

A	B	C
a	b	$\perp$
a	$\perp$	c

(ii)

Figure 2: Some problematic relations

If we consider these sets of elements in a domain as sets of descriptions then we would like to order the sets themselves by how good they describe sets of real-world objects, but how? The study of the semantics of non-determinism, which attempts to describe the behavior of sets of processes, provides us with some answers. However, we must first decide whether we are prepared to work arbitrary sets, or whether some restrictions are needed.

Given a domain $(\mathcal{D}, \sqsubseteq)$, a set $S \subset \mathcal{D}$ is a *co-chain* if no member of S is greater than any other member of S, i.e. $\forall x, y \in S. x \sqsupseteq y$ implies $x = y$. If $S \subset \mathcal{D}$ has the property that any two members of S are *inconsistent*, i.e. they do not have a defined join, then we shall call S *independent*. Note that an independent set is necessarily a co-chain.

First-normal-form relations are independent sets. If, however, we admit null values in relations by relaxing the condition $\{l \mid d(l) \neq \perp\} = R$ of first-normal-relation to $\{l \mid d(l) \neq \perp\} \subseteq R$, we have to decide whether structures such as (i) or (ii) of figure 2 are valid relations. (i) fails to be a co-chain because $\{A \Rightarrow a\} \sqsubseteq \{A \Rightarrow a;\ B \Rightarrow b\}$, and (ii) fails to be independent because $\{A \Rightarrow a;\ B \Rightarrow b\} \sqcup \{A \Rightarrow a;\ C \Rightarrow c\}$ is defined.

In what follows we shall assume that database sets are finite co-chains and we shall use the words *finite co-chain* and *relation* interchangeably. Using our simple notion of database semantics, we might justify this assumption by saying that if d_1 and d_2 are descriptions with d_2 a better description than d_1 then d_1 is redundant and can be eliminated from the database. This is equivalent to saying that for all pairs d_1, d_2 in S neither $[\![d_1]\!] \subseteq [\![d_2]\!]$ nor $[\![d_1]\!] \subseteq [\![d_2]\!]$. Whether or not this justification is reasonable depends on the intended semantics of the operations on co-chains which, in turn, depends on the circumstances in which they are used. See [28] for a more detailed

	Dept	Office
R_1	*'Mktg'*	275
	'Sales'	147

	Name	Dept	Office
R_2	*'K. Smith'*	*'Mktg'*	275
	'L. Jones'	*'Mktg'*	275

	Name	Dept	Office
	'K. Smith'	*'Mktg'*	275
R_3	*'L. Jones'*	*'Mktg'*	275
	'J. Doe'	*'Sales'*	147
	'M. Blake'	*'Sales'*	147

$$R_1 \sqsubseteq^{\sharp} R_2 \quad R_2 \sqsubseteq^{\flat} R_3 \quad R_1 \sqsubseteq^{\natural} R_3$$

Figure 3: Examples of the three orderings

examination of the semantics of relational operations. Independence means that no two descriptions in S can describe the same real-world object, i.e. $[\![d_1]\!] \cap [\![d_2]\!] = \emptyset$. We shall need to discuss independent sets when we generalize the notion of schemes. We shall use $\mathcal{C}_{\mathcal{D}}$ to refer to the set of finite co-chains in $\mathcal{D}$ and $\mathcal{I}_{\mathcal{D}}$ for the set of finite independent sets.

To return to the problem of finding orderings on sets the study of the semantics of non-determinism provides us with three orderings[1]:

$$\begin{array}{lll} A \sqsubseteq^{\flat} B & \text{if} & \forall a \in A \exists b \in B . a \sqsubseteq b \\ A \sqsubseteq^{\sharp} B & \text{if} & \forall b \in B \exists a \in A . a \sqsubseteq b \\ A \sqsubseteq^{\natural} B & \text{if} & A \sqsubseteq^{\flat} B \text{ and } A \sqsubseteq^{\sharp} B \end{array}$$

respectively called the Hoare, Smyth, and Egli-Milner ordering. Figure 3 shows examples of these orderings in first-normal-form relations.

For arbitrary sets, these are not orderings; they are pre-orderings and orderings are derived by taking equivalence classes. However, in each case there are canonical representatives for each equivalence class:

Lemma 1 *Let P be a partial order. Then the following is true for all subsets A and B of P.*

[1] This melodious notation was suggested to us by Carl Gunter.

(i) $A =^\flat \downarrow A$.

(ii) $A \sqsubseteq^\flat B \Leftrightarrow \downarrow A \subseteq \downarrow B$.

(iii) $A =^\sharp \uparrow A$.

(iv) $A \sqsubseteq^\sharp B \Leftrightarrow \downarrow A \subseteq \uparrow B$.

(v) $A =^\natural \uparrow A \cap \downarrow A$. □

So in reasoning about these orderings it is helpful to think in terms of lower sets, upper sets, and order-convex sets, respectively. We said before that we want to model database sets (or relations) as finite co-chains in our domains. Since databases tend to get bigger and bigger during their existence one might think that the Hoare ordering is the most natural for them. However, viewed as approximations of sets of real world objects it is the Smyth ordering which corresponds to this semantics. We regard it as a strength of our approach that it allows to formalize different intuitions about databases. The mathematics is nice in each case:

Lemma 2 *If $\mathcal{D}$ is a domain then $(\mathcal{C}_\mathcal{D}, \sqsubseteq^\flat)$ and $(\mathcal{C}_\mathcal{D}, \sqsubseteq^\sharp)$ are distributive lattices with bottom element. $(\mathcal{C}_\mathcal{D}, \sqsubseteq^\sharp)$ also has a top element, namely, the empty co-chain.*

Proof. Given two finite co-chains S_1 and S_2, it is clear how the sup and inf are found for each of the two orderings:

$S_1 \sqcup^\flat S_2 =$ the maximal elements of $\downarrow S_1 \cup \downarrow S_2 =$ the maximal elements of $S_1 \cup S_2$.

$S_1 \sqcap^\flat S_2 =$ the maximal elements of $\downarrow S_1 \cap \downarrow S_2 \subseteq S_1 \sqcap S_2 = \{s_1 \sqcap s_2 \mid s_1 \in S_1, s_2 \in S_2\}$.

$S_1 \sqcup^\sharp S_2 =$ the minimal elements of $\uparrow S_1 \cap \uparrow S_2 \subseteq S_1 \sqcap S_2$.

$S_1 \sqcap^\sharp S_2 =$ the minimal elements of $\uparrow S_1 \cup \uparrow S_2 \subseteq S_1 \cup S_2$.

Distributivity follows because we can embed $\mathcal{C}_\mathcal{D}$ in the distributive lattice of all lower (upper) sets in $\mathcal{D}$. □

We wish to remark that these lattices are not complete: Neither $(\mathcal{C}_\mathcal{D}, \sqsubseteq^\flat)$ nor $(\mathcal{C}_\mathcal{D}, \sqsubseteq^\sharp)$ contain sups for directed subsets. If we want completeness then we have to take certain computability considerations into account which translate into topological restrictions on infinite subsets of a domain. We have no need to pursue this theme further but note that sup and inf in both orderings are defined for any set of subsets of a domain. They may not be representable by their subset of minimal or maximal elements, however.

It the space of finite co-chains with the three orderings in which we represent various operations on database sets, some of which will emerge as generalizations of relational operations. We also mention that these ordered spaces are not the same as powerdomains in the programming language literature [33, 45], where the ordered spaces of sets are constructed in such a way that they are themselves domains. True powerdomain constructions are not needed until we discuss higher-order relations, where a tuple can itself contain a set as an attribute value. We shall discuss how our presentation of database sets can also contain these higher-order values in section 7, but for the time being we shall exploit the representation of database sets in the space of finite co-chains.

There is an immediate connection with relational algebra that indicates the importance of these orderings.

Theorem 1 *Interpreting relations as finite co-chains A, B in $\mathcal{L} \to \mathcal{V}_\bot$, $A \sqcup^\sharp B$ is the natural join of A and B. If a least upper bound for A, B exists in $\sqsubseteq^\natural$ then it is a lossless join.*

This statement is actually more of a definition than a result. We can only prove it in the case of first-normal-form relations, for it is only then that we have accepted definitions for the various joins. Given relation schemes (sets of attribute names) $R_1, R_2 \subseteq \mathcal{L}$ and relation instances r_1, r_2, let r_1', r_2' and r_3' be the interpretations of r_1, r_2 and $r_1 \bowtie r_2$ in $\mathcal{L} \to \mathcal{V}_\bot$. Suppose $t \in r_3'$, then by the conventional definition of natural join, there are $t_1 \in r_1'$ such that $t(l) = t_1(l)$ for all $l \in R_1$ and $t_2 \in r_2'$ such that $t(l) = t_2(l)$ for all $l \in R_2$. By the definition of the interpretation, $t_1(l) \neq \bot$ iff $l \in R_1, t_2(l) \neq \bot$ iff $l \in R_2$. This implies $t_1 \sqsubseteq t$ and $t_2 \sqsubseteq t$ and clearly t is minimal

$$
\begin{array}{lll}
r_1 & = \{ & \{Name\Rightarrow \textit{'J. Doe'} \quad ;Status\Rightarrow\{\ Student\text{-}status\Rightarrow\textit{'Graduate'}\}\} \\
 & & \{Name\Rightarrow \textit{'M. Blake'};Status\Rightarrow\{\ Student\text{-}status\Rightarrow\textit{'Undergraduate'}\}\}\} \\
r_2 & = \{ & \{Name\Rightarrow\textit{'J. Doe'};\quad Status\Rightarrow\{\ Employee\text{-}status\Rightarrow\textit{'TA'}\}\} \\
 & & \{Name\Rightarrow\textit{'L. Jones'};\ Status\Rightarrow\{\ Employee\text{-}status\Rightarrow\textit{'Facalty'}\}\}\} \\
r_1 \sqcup^\sharp r_2 & = \{ & \{Name\Rightarrow\textit{'J. Doe'}; \\
 & & \quad Status\Rightarrow\{\ Student\text{-}status\Rightarrow\textit{'Graduate'};Employee\text{-}status\Rightarrow\textit{'TA'}\}\}\}
\end{array}
$$

Figure 4: Natural join in "nested" records

with respect to this property. Therefore $t \in r_1' \sqcup^\sharp r_2'$. Conversely suppose $t \in r_1' \sqcup^\sharp r_2'$. There must exist $t_1 \in r_1, t_2 \in r_2$ such that $t_1 \sqsubseteq t$ and $t_2 \sqsubseteq t$. Since $\mathcal{V}_\perp$ is flat this implies that $t(l_1) = t_1(l_1)$ when $l_1 \in R_1$ and $t(l_2) = t_2(l_2)$ when $l_2 \in R_2$. By the minimality of t with respect to $\sqsubseteq$, $t(l) = \perp$ iff $l \notin R_1 \cup R_2$. Hence $t \in r_3'$. See [28] for a discussion of the semantics of lossless join and the proof of the second part of this result. $\Box$

The importance of this result is that it provides a generalization of natural join to sets of values in arbitrary domains. Figure 4 shows an example of natural join in nested records.

A more intuitive way of thinking of these results is to view the natural join as the appropriate operation when two sets of database descriptions "over-approximate" some desired set in the real world. Suppose, for example, that we want to find the set of TEACHING-FELLOWS, but we only have available database sets describing EMPLOYEES and STUDENTS. Both of these over-approximate our desired set (any teaching fellow is both an employee and a student) and so the appropriate operation to achieve a better approximation to TEACHING FELLOWS is to take the natural join of EMPLOYEES and STUDENTS.

The partial ordering $\sqsubseteq^\natural$ does not give rise to least upper bounds when applied to co-chains. However, if two database sets have a least upper bound in $\sqsubseteq^\natural$, then any real world set that is "exactly" described by (i.e. above in $\sqsubseteq^\natural$) the two database sets is also "exactly" described by the least upper bound. Since a least upper bound in $\sqsubseteq^\natural$ is also a least upper bound in $\sqsubseteq^\sharp$, if $\sqcup^\natural$ exists then the natural join is the lossless join. Traditionally the lossless join condition is stated operationally, in terms of projections; from this we see that it has a simple denotational interpretation.

We might also ask whether $\sqcup^\flat$ corresponds to anything in the relational algebra. $S_1 \sqcup^\flat S_2$ is simply the set of maximal elements in $S_1 \cup S_2$ and is awkward to deal with in relational algebra as it generally requires the introduction of null values. However we shall make some use of this operator later. If we are prepared to introduce null values, then $\sqcup^\flat$ is what [35] calls the "null union", and $S_1 \sqcup^\flat (S_1 \sqcup^\sharp S_2) \sqcup^\flat S_2$ is what is sometimes called the *outer join*. Merret [26] describes this operation and also the "left-wing" and "right-wing" operations, which are $S_1 \sqcup^\flat (S_1 \sqcup^\sharp S_2)$ and $(S_1 \sqcup^\sharp S_2) \sqcup^\flat S_2$ respectively.

In some cases these operations preserve independence:

Lemma 3 *If S_1 and S_2 are independent, so are $S_1 \sqcup^\sharp S_2$, $S_1 \sqcup^\flat (S_1 \sqcup^\sharp S_2) \sqcup^\flat S_2$, $S_1 \sqcup^\flat (S_1 \sqcup^\sharp S_2)$ and $(S_1 \sqcup^\sharp S_2) \sqcup^\flat S_2$.* $\Box$

However, the other operators ($\sqcap^\flat, \sqcap^\sharp$ and $\sqcup^\flat$) do not, in general, carry independent sets into independent sets.

We should also note that the co-chain $S_1 \sqcap^\sharp S_2$ is the set of minimal elements of $S_1 \cup S_2$. When $S_1 \cup S_2$ is a co-chain, $S_1 \sqcap^\sharp S_2 = S_1 \sqcup^\flat S_2$. The operator $\sqcap^\flat$ is, as we shall see in the next section, a general form of projection.

In order to conform to traditional notation, we shall generally replace the symbol $\sqcup^\sharp$ by what is conventionally used in databases, $\bowtie$.

4 Projection

The main point of the previous section is that we are able to define various joins without reference to the special structure of relations. In particular, we do not require any notion of sets of column names (or schemes as they are called in the relational database literature [24, 46]) in order to characterize natural join. Projection, however, makes explicit mention of a scheme. For example $\{Name,\ Office\}$ is a scheme and the projection $\pi_{\{\text{Name, Office}\}}(R)$ where R is the relation shown in Figure 1 is

$$\{\{Name \Rightarrow \text{'J. Doe'}; Office \Rightarrow 147\},$$
$$\{Name \Rightarrow \text{'K. Smith'};\ Office \Rightarrow 275\}\}\ .$$

If, therefore, we are to carry further the idea of casting relational algebra in the theory of domains, we need to generalize the notion of relational schemes and projection.

We have essentially two options: the first is to look at what properties are desired of the projection function itself; the second is to identify schemes with some set of elements in the underlying domain $\mathcal{D}$. The second approach is motivated by the idea that a set of column names gives rise to a smaller universe of descriptions. For example, we might say that the relational scheme $\{Name,\ Office\}$ denotes the set of all descriptions (functions) of the form $\{\{Name \Rightarrow v,\ Office \Rightarrow w\} \mid v, w \in \mathcal{V}\}$. The course we shall follow is to look at both possibilities with the goal of finding some characterization that is natural in the sense that it admits some natural algebra over the set of schemes. This is essential if we are to generalize ideas about functional dependencies which are usually cast in the boolean algebra of sets. However, the authors should admit here that the generalization of schemes that we are going to provide, while it arises from extremely natural conditions and captures a number of relational database constructs, may require further refinement if it is to be used for all of relational database theory. We do not know, for example, whether we can represent multi-valued dependencies using our characterization.

We start from the observation that in relational databases we can say what projection means for a single tuple. It is simply the function that throws away certain fields from a tuple or record. More generally, we can think of projection as a function $p \in \mathcal{D} \to \mathcal{D}$ that is decreasing, idempotent and monotone, i.e. for all $x, y \in \mathcal{D}$, $p(x) \sqsubseteq x$, $p(p(x)) = p(x)$, and $p(x) \sqsubseteq p(y)$ whenever $x \sqsubseteq y$. Computability of a projection is reflected in the property of preserving directed sups: $p(\bigsqcup_{i \in I} x_i) = \bigsqcup_{i \in I} p(x_i)$. Such functions are also known as projections in domain theory, and it is clear that a (relational) projection onto a set of column names satisfies these conditions.

Projections are completely determined by their image:

Lemma 4 *Let $\mathcal{D}$ be a domain and p, p' be projections on $\mathcal{D}$.*

(i) $p(x) = \bigsqcup\{y \mid p(y) = y \sqsubseteq x\}$.

(ii) $p \sqsubseteq p' \Leftrightarrow im(p) \subseteq im(p') \Leftrightarrow p \circ p' = p' \circ p = p$.

(iii) p preserves inf's of nonempty sets.

(iv) $im(p)$ is closed under existing sup's. $\square$

We feel that arbitrary projections as defined above do have a significance in modeling databases domain theoretically. In this paper, however, we shall concentrate on a more restricted notion of projection which we shall develop in two steps.

In the case of a relational domain $\mathcal{L} \to \mathcal{V}_\perp$, restricting the set of labels to some subset L of $\mathcal{L}$ gives rise to a downward closed subset of $\mathcal{L} \to \mathcal{V}_\perp$, namely the set of all functions s for which $s(l) = \perp$ if $l \notin L$.

Definition. Let $\mathcal{D}$ be a domain. A *strong ideal* in $\mathcal{D}$ is a downward closed subset A of $\mathcal{D}$ which is closed under existing joins. By p_A we denote the unique projection on $\mathcal{D}$ with image A.

Projections onto strong ideals enjoy several desirable properties:

Lemma 5 *(i) Let A be a strong ideal in a domain $\mathcal{D}$ and let y be an element of A above $p_A(x)$ for some $x \in \mathcal{D}$. Then $p_A(x) = x \sqcap y$.*

(ii) If $\mathcal{D}$ is distributive then p_A preserves all existing sup's.

Proof. (i) By definition we have $p_A(x) \sqsubseteq x$ and $p_A(x) \sqsubseteq y$ so $p_A(x) \sqsubseteq x \sqcap y$. On the other hand, $x \sqcap y$ is an element of A below x and $p_A(x)$ is the sup of all those elements, hence $x \sqcap y \sqsubseteq p_A(x)$.

(ii) Suppose $x \sqcup y$ exists. Then $p_A(x) \sqcup p_A(y) = (x \sqcap p_A(x \sqcup y)) \sqcup (y \sqcap p_A(x \sqcup y))$ (by (i)) $= p_A(x \sqcup y) \sqcap (x \sqcup y)$ (distributivity) $= p_A(x \sqcup y)$. The sup of any set is equal to the directed sup of its finite subsets. Our projection preserves both kinds of sups, hence arbitrary sups. $\square$

The intersection of an arbitrary set of strong ideals is again a strong ideal. This immediately gives us the following

Theorem 2 *The set $(\mathcal{SI}_\mathcal{D}, \subseteq)$ of strong ideals on a (distributive) domain $\mathcal{D}$ is a (distributive) algebraic lattice.* $\square$

The second condition on projections we want to consider here is also easily motivated by the example of flat record structures $\mathcal{L} \to \mathcal{V}_\perp$. Suppose we project onto records with labels from some subset L of $\mathcal{L}$ and we find that a record s is projected onto $p_L(s)$ below some $s' \in L \to \mathcal{V}_\perp$. This means that $p_L(s)$ contains null values for some labels from L and can be updated using the corresponding entries of s'. It is clear, then, that s itself can be updated, resulting in the record $s \sqcup s'$. We incorporate this property in our model as follows.

Definition. A strong ideal A in a domain $\mathcal{D}$ satisfies the *slide condition* if $\forall x \in \mathcal{D}. \forall y \in A.(p_A(x) \sqsubseteq y \Rightarrow x \sqcup y$ exists). A co-chain S in $\mathcal{D}$ is a *scheme* if $\downarrow S$ is a strong ideal which satisfies the slide condition. The corresponding projection we denote by p_S (instead of $p_{\downarrow S}$).

We first note that projections defined by schemes fit in with our proposed semantics:

Theorem 3 *A strong ideal A on a domain $\mathcal{D}$ is generated by a scheme if and only if $\forall x \in \mathcal{D}. p_A[\![x]\!]_\mathcal{D} = [\![p_A(x)]\!]_A$.*

Proof. "$\Rightarrow$" Let x be maximal in $\mathcal{D}$ and suppose $p_A(x)$ is not maximal in A, that is, $p_A(x) \sqsubset y \in A$. By the slide condition, $x \sqcup y$ exists and since x is maximal, $x \sqcup y = x$ and $p_A(x) \sqsupseteq y$. Contradiction. Given any $x \in \mathcal{D}$ and any $y \in [\![p_A(x)]\!]_A$, the sup of x and y exists and is below some maximal element z of $\mathcal{D}$. Clearly, $p_A(z) = y$ by the maximality of y.

"$\Leftarrow$" Given $x \in \mathcal{D}$ and $y \in A$, $y \sqsupseteq p_A(x)$, let y' be an element of A_{max} above y. This element must be in the image of $p_A[\![x]\!]_\mathcal{D}$, that is, there exists an element z of $\mathcal{D}_{max} \cap \uparrow x$ which is mapped onto y'. Therefore x and y are bounded and $x \sqcup y$ exists. $\Box$

In section 7.1 we shall further substantiate our claim that schemes properly generalize the notion of schemes in relational database theory by showing that schemes in the domain $\mathcal{L} \to \mathcal{V}_\perp$of flat record structures correspond exactly to the subsets of $\mathcal{L}$.

Lemma 6 *Let $\mathcal{D}$ be distributive domain.*

(i) If A and B are strong ideals generated by schemes then so is $A \sqcup B = \{a \sqcup b \mid a \in A, b \in B\}$.

(ii) If A and B are strong ideals generated by schemes then so is $A \cap B$.

(iii) If $(A_i)_{i\in I}$ is any set of strong ideals generated by schemes then so is $\bigsqcup_{i\in I} A_i = \{\bigsqcup_{i\in I} a_i \mid a_i \in A_i\}$.

(iv) If A, B are schemes then so is $A \sqcup B = \{a \sqcup b \mid a \in A, b \in B\}$.

(v) If $(A_I)_{i\in I}$ is any set of schemes then so is $\bigsqcup_{i\in I} A_i = \{\bigsqcup_{i\in I} a_i \mid a_i \in A_i\}$.

Proof. (i) $A \sqcup B$ is downward closed: $x \sqsubseteq a \sqcup b$ implies $x = x \sqcap (a \sqcup b) = (x \sqcap a) \sqcup (x \sqcap b)$ and $x \sqcap a$ is in A and $x \sqcap b$ is in B. If M is a bounded subset of $A \sqcup B$ then $M_A = \{a \in A \mid \exists b \in B. a \sqcup b \in M\}$ and $M_B = \{b \in B \mid \exists a \in A. a \sqcup b \in M\}$ are bounded and $\bigsqcup M_A = m_A \in A$ and $\bigsqcup M_B = m_B \in B$. Hence $\bigsqcup M = m_A \sqcup m_B$ is in $A \sqcup B$. As for the slide condition, assume that x is an element of $\mathcal{D}$ and that $p_{A\sqcup B}(x)$ is below some $a \sqcup b$. Because $p_A(a \sqcup b) \sqsupseteq a$ and $p_B(a \sqcup b) \sqsupseteq b$ we may assume that $a = p_A(a \sqcup b)$ and $b = p_B(a \sqcup b)$. We can then calculate: $p_A(x) = p_A(p_{A\sqcup B}(x)) \sqsubseteq p_A(a \sqcup b) = a$ and similarly $p_B(x) \sqsubseteq b$. Since A satisfies the slide condition, the sup of a and x exists and by Lemma 5 $p_B(a \sqcup x) = p_B(a) \sqcup p_B(x) \sqsubseteq p_B(a \sqcup b) \sqcup b = b$. Using the slide condition for B we find that the sup of $a \sqcup x$ and b must exist. This proves the slide condition for $A \sqcup B$.

(ii) $A \sqcap B$ is clearly a strong ideal. The slide condition is seen to hold by the following argument. If $p_{A\sqcap B}(x)$ is below $y \in A \sqcap B$ then because of $p_{A\sqcap B} = p_A \circ p_B$, p_A maps $p_B(x)$ below y. Hence $y \sqcup p_B(x)$ exists and is an element of B. Using the slide condition for B we see that $y \sqcup x$ exists.

(iii) If I is empty then $\bigsqcup_{i\in I} A_i$ equals $\{\bot\}$ which is a scheme. If I is infinite then we may think of I as the directed union of its finite subsets. From part (i) we already know how to construct the sup of a finite set of strong ideals, so it remains to consider directed collections. Assume, therefore, that I is directed and that $A_i \subseteq A_j$ whenever $i \leq j$. Given an element x of $A = \bigsqcup_{i\in I} A_i$ first note that $x = \bigsqcup_{i\in I} p_{A_i}(x)$ and that this join is directed. It is clear that A is a strong ideal. Let X be any element of $\mathcal{D}$ and let $y \in A$ be above $p_A(x)$. Then for each $i \in I$, $p_{A_i}(y)$ is above $p_{A_i}(p_A(x))$. So the sup $z_i = p_{A_i}(y) \sqcup x$ exists and the directed sup of all z_i gives us the sup of y and x. Therefore A satisfies the slide condition.

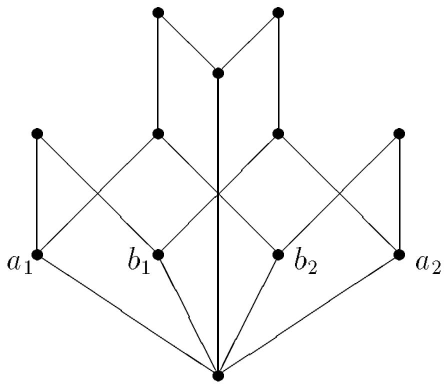

Figure 5: A domain where the join of two schemes cannot be calculated pointwise.

(iv) By (i) it remains to show that $A \sqcup B$ is an independent set. Indeed, if x is above $a_1 \sqcup b_1$ and $a_2 \sqcup b_2$ it follows that $a_1 = a_2$ and $b_1 = b_2$ because both A and B are independent sets.

(v) Same proof as for (iv). $\Box$

Distributivity is essential for Part (i) of this lemma, as the example shown in Figure 5 demonstrates. There the pointwise sup of the schemes $\{a_1, a_2, \bot\}$ and $\{b_1, b_2, \bot\}$ does not satisfy the slide condition.

We plan to present a deeper investigation into the mathematics of schemes in a later paper, but mention that ideals generated by schemes form a complete lattice:

Theorem 4 *If $\mathcal{D}$ is a distributive domain then $(\mathcal{S}_{\mathcal{D}}, \sqsubseteq^\flat)$ is a distributive complete lattice.* $\Box$

In the remainder of this section we shall work the generating co-chain of an ideal, that is, with schemes. It turns out that the ordering $\subseteq$ on strong ideals is replaced by the Egli-Milner ordering $\sqsubseteq^\natural$ on schemes.

Theorem 5 *If A and B are schemes on a domain $\mathcal{D}$ then $A \sqsubseteq^\flat B$ if and only if $A \sqsubseteq^\sharp B$.*

Proof. "$\Rightarrow$" Let x be an element of B and let y be maximal above x in $\mathcal{D}$. Then $p_A(y) = p_A(p_B(y)) = p_A(x)$ and therefore $p_A(x)$ is maximal in $\downarrow A$ which means that it is contained in A.

"$\Leftarrow$" If x is an element of A, let y be maximal in $\mathcal{D}$ above x. Since $x = p_A(y) \sqsupseteq p_A(p_B(y))$, the sup of x and $p_B(y)$ exists. $p_B(y)$ is maximal in $\downarrow B$ and because of $A \sqsubseteq^\sharp B$ it is above some $x' \in A$. The set $\{x, x'\}$ is bounded by $x \sqcup p_B(y)$ which is only possible if $x = x'$. Hence $x \sqsubseteq p_B(y)$ and $A \sqsubseteq^\flat B$. $\Box$

So far we have discussed the projection of individual elements ("records") into strong ideals. We shall now proceed to discuss the projection of relations, that is, finite co-chains. The obvious choice, namely, to apply the projection pointwise, has its particular applications. However, we might not get a co-chain as the image. Throwing away redundant information means in our case to keep only the maximal elements of the image.

Definition. Let $\mathcal{D}$ be a domain and A be a scheme in $\mathcal{D}$.

(i) The function $\pi_A: \mathcal{C}_\mathcal{D} \to \mathcal{C}_\mathcal{D}$ is defined by $\pi_A(R) = \{x \in p_A(R) \mid x \text{ maximal in } p_A(R)\}$.

(ii) If $R \in \mathcal{C}_\mathcal{D}$ is a subset of A, we shall call R an *instance* of A.

(iii) If $R \in \mathcal{C}_\mathcal{D}$ is a subset of $\downarrow A$, we shall call it a *subinstance* of A.

Theorem 6 *Let A, B be schemes in a distributive domain $\mathcal{D}$.*

(i) If R is an instance of A then $R \sqsubseteq^\flat A$ and $A \sqsubseteq^\sharp R$.

(ii) If R is an (sub-)instance of A and S is an (sub-)instance of B then $R \bowtie S$ and $R \sqcup^\natural S$ (if it exists) are (sub-)instances of $A \sqcup B$ and $R \sqcup^\flat S$ is a subinstance of $A \sqcup B$.

(iii) If R is an instance of A then $\pi_B(R)$ is a subinstance of B.

(iv) If R is a finite co-chain in $\mathcal{D}$ then $p_A(R) \bowtie R = R$.

(v) If R is a finite co-chain in $\mathcal{D}$ then $\pi_A(R) \bowtie R \sqsupseteq^\sharp R$.

(vi) If R is a finite independent set in $\mathcal{D}$ then $\pi_A(R) \bowtie R \sqsupseteq^\natural R$.

Proof. Of these only (vi) is nontrivial. One half of the Egli-Milner ordering follows from (v). As for the "Hoare"-part we can copy the corresponding proof of Theorem 5. □

Let us recapitulate the development of our theory so far. We have exhibited a general structure which may take the place of attribute value sets in relational database, namely distributive Scott-domains. We proposed to model relations as finite co-chains in these domains. In Lemma 2 we have shown that relations form a distributive lattice under two natural orderings which correspond to the two intuitions one might have about a relation: One being that a relation gives information about a part of a set of real world objects, the other being that a relation approximates every element of a set of real world objects. We then proceeded to model the notions of scheme and projection and found (Theorem 4) that schemes form a distributive complete lattice. This says that the set of schemes is nearly a powerset and allows to interpret intuitionistic logic in it. Along the way we have indicated the possibilities for fine tuning in this model: Using independent sets instead of co-chains or generalizing schemes to strong ideals. We shall now go on to test our theory in two fields, that of functional dependencies and that of universal relations.

5 Functional Dependencies

We start again with the familiar example of a relational database. Given some set of functional dependencies and given a set A of attribute names one can use Armstrong's Axioms in order to produce a set $A' \supseteq A$ which contains all attribute names depending on A. In our domain theoretic setting we may view this process as a function on the lattice of schemes, which is monotone, idempotent and increasing. These functions are the exact counterpart of projections as discussed in the previous section.

Definition. A *closure* on a domain $\mathcal{D}$ is a monotone function $f : \mathcal{D} \to \mathcal{D}$, such that $f \circ f = f \sqsupseteq id_{\mathcal{D}}$.

Lemma 7 *Let $\mathcal{D}$ be a domain and f, f' be closures on $\mathcal{D}$.*

(i) $f(x) = \sqcap\{y \mid f(y) = y \sqsupseteq x\}$.

(ii) $f \sqsupseteq f' \Leftrightarrow im(f) \subseteq im(f') \Leftrightarrow f \circ f' = f' \circ f = f$.

(iii) f *preserves all existing sup's.*

(iv) $im(f)$ *is closed under nonempty inf's.* □

This is, of course, the exact dual of Lemma 4. Note that because of part (iii), closures are always continuous.

Given a function $f : \mathcal{D} \to \mathcal{D}$, we can define a relation $\tilde{f} \subseteq \mathcal{D} \times \mathcal{D}$ by $\tilde{f} = \{(x, y) \mid y \sqsubseteq f(x)\}$ and obtain an immediate connection with Armstrong's Axioms.

Theorem 7 *If* f *is a closure in* $\mathcal{D} \to \mathcal{D}$, $\tilde{f}$ *satisfies*

(a) $\forall x, y \in \mathcal{D}$ *if* $x \sqsupseteq y$ *then* $(x, y) \in \tilde{f}$,

(b) *if* $S \subseteq \mathcal{D}$ *is such that* $\forall y \in S.(x, y) \in \tilde{f}$ *then* $\sqcup S$ *exists and* $(x, \sqcup S) \in \tilde{f}$, *and*

(c) $\forall x, y, z \in \mathcal{D}.(x, y) \in \tilde{f}$ *and* $(y, z) \in \tilde{f} \Rightarrow (x, z) \in \tilde{f}$.

When $\mathcal{D}$ *is finite* (b) *may be replaced by*

(b′) *for* $x, y, w \in \mathcal{D}$ *if* $(x, y) \in \tilde{f}$ *and* $x \sqcup w$ *exists then* $w \sqcup y$ *exists, and* $(w \sqcup x, w \sqcup y) \in \tilde{f}$.

Conversely, suppose $\tilde{f} \subseteq \mathcal{D} \times \mathcal{D}$ *satisfies* (a), (b) *or* (b') *as appropriate, and* (c) *above, and define* $f : \mathcal{D} \to \mathcal{D}$ *by* $f(x) = \sqcup\{y \mid (x, y) \in \tilde{f}\}$. *Then* f *is a closure.*

From which (a), (b′) and (c) are immediately seen to be generalizations of Armstrong's Axioms. Before discussing the connection, we should prove this result. (a) follows immediately from the definition of a closure since if $y \sqsubseteq x$, then $y \sqsubseteq f(x)$ and $(x, y) \in \tilde{f}$. (b) is also immediate because $f(x)$ must be a bound for S, therefore $\sqcup S$ exists and $\sqcup S \sqsubseteq f(x)$. To show (c), if $(x, y) \in \tilde{f}$ then $y \sqsubseteq f(x)$ and by monotonicity and idempotence $f(y) \sqsubseteq f(x)$. The conditions also imply $z \sqsubseteq f(y)$. Combining these last two inequalities we have $z \sqsubseteq f(x)$, i.e. $(x, z) \in \tilde{f}$. Conversely, we first note that condition (b) implies that $\sqcup\{y \mid (x, y) \in \tilde{f}\}$ exists and f

is well defined. If $x_1 \sqsubseteq x_2$ and $(x_1, y) \in \tilde{f}$ then $(x_2, y) \in \tilde{f}$ by (a) and (c) so that $\{y \mid (x_1, y) \in \tilde{f}\} \subseteq \{y \mid (x_2, y) \in \tilde{f}\}$, and hence $f(x_1) \sqsubseteq f(x_2)$ guaranteeing monotonicity. By (a) $(x, x) \in \tilde{f}$, so $f(x) \sqsupseteq x$. Finally, by (b) $(x, \sqcup\{y \mid (x, y) \in \tilde{f}\}) \in \tilde{f}$, and so $(x, f(x)) \in \tilde{f}$; similarly $(f(x), f(f(x))) \in f$. Using (c), $(x, f(f(x)) \in \tilde{f}$ and so $f(f(x)) \sqsubseteq f(x)$. But we have just shown that f is increasing. Hence $f(f(x)) = f(x)$.

Suppose (a), (b), (c) hold and that $(x, y) \in \tilde{f}$. For any $w \in \mathcal{D}$, $(w \sqcup x, w) \in \tilde{f}$ and $(w \sqcup x, x) \in \tilde{f}$ by (a), and by (c) $(w \sqcup x, y) \in \tilde{f}$. Therefore, by (b) $(w \sqcup x, w \sqcup y) \in \tilde{f}$. Conversely, assume $\mathcal{D}$ finite. First note that, by putting $w = x$ in (b$'$) we have $x \sqcup y$ exists. Suppose $\forall y \in S.(x, y) \in \tilde{f}$. If S has just two members, y_1, y_2 then $(x, x \sqcup y_1) \in \tilde{f}$ by (b$'$) and $(x, x \sqcup y_1 \sqcup y_2) \in \tilde{f}$ by (c), therefore $y_1 \sqcup y_2$ exists. Using (c) and (a) we get $(x, y_1 \sqcup y_2) \in \tilde{f}$, i.e. $(x, \sqcup S) \in \tilde{f}$. By induction, we can repeat this argument to derive (b) for any finite S. $\Box$

Armstrong's Axioms are precisely (a), (b$'$), (c) when applied to the lattice of subsets of the set of attribute names. Related characterizations of Armstrong's Axioms in a lattice-theoretic setting have been given by [20]. It is also interesting that in Scott's information systems [42] functions on domains are defined by a similar device of taking approximating relations.

We now connect this abstract notion of a functional dependency with our earlier semantics in which sets of attribute names are represented by schemes. A relation satisfies a functional dependency $A \to B$ if any two tuples that agree on the attribute names A agree on the attribute names B. Another way of stating this is to follow [16] and say that a relation r satisfies $A \to B$ if the partition on r induced by A (i.e. the equivalence relation induced on the tuples by equality on A) is finer than the partition induced by B. In the standard theory there are no null values allowed in places corresponding to attributes from $A \cup B$. We keep this strong interpretation of satisfaction.

Definition. Let A, B be schemes in a domain $\mathcal{D}$. A relation $R \in \mathcal{C}_{\mathcal{D}}$ *satisfies* the functional dependency $A \to B$ if $R \sqsupseteq^\sharp A$ and $R \sqsupseteq^\sharp B$ and if $p_A(x) = p_A(y)$ implies $p_B(x) = p_B(y)$ for all $x, y \in R$.

Theorem 8 *For relations in distributive domains Armstrong's Axioms are consistent and complete.*

Proof. Given a relation R in a distributive domain $\mathcal{D}$ and given a scheme $A \sqsubseteq^\sharp R$ it is clear that R satisfies $A \to A$. If S is a collection of schemes and R satisfies $A \to B$ for all $B \in S$ and some scheme A, then S is bounded by R in the Smyth ordering. We claim that the sup of S is also below R: If x is an element of R and B is a scheme contained in S then x is above some element x_B of B. Therefore x bounds the set $X = \{x_B \mid B \in S\}$. The sup of X is an element of $\bigsqcup S$ by Lemma 6, (v) and is below x. This proves $\bigsqcup S \sqsubseteq^\sharp R$. Assume, then, that $p_A(x) = p_A(y)$. By assumption we know that $p_B(x) = p_B(y)$ for all $B \in S$. Hence we also have $p_{\sqcup S}(x) = \bigsqcup_{B \in S} p_B(x) = \bigsqcup_{B \in S} p_B(y) = p_{\sqcup S}(y)$, which proves $A \to \bigsqcup S$.

It is clear that transitivity holds. This proves that Armstrong's Axioms are correct with respect to our definition of satisfaction.

Completeness is trivial because we have more models available than in the relational case. □

It is an immediate consequence of the preceding theorem and Theorem 7 that a relation $R \in \mathcal{C}_\mathcal{D}$ induces a closure f on the lattice of schemes with the property $f(A) \sqsupseteq B$ if and only if R satisfies $A \to B$.

Our definition of satisfaction of a dependency requires that the relation under consideration contains no partial information. If a relation does contain partial elements, a different concept is called for.

Definition. Let A, B be schemes in a domain $\mathcal{D}$. A relation $R \in \mathcal{C}_\mathcal{D}$ is *consistent* with the functional dependency $A \to B$ if there is a relation $R' \sqsupseteq^\natural R$ which satisfies $A \to B$.

This is natural enough. However, in a practical instance consistency may be hard to check. We therefore introduce a weaker notion of consistency with a more operational flavor. Given a scheme (or any independent set) A and a relation R then A induces a partial equivalence relation $\sim_A$ on R: $x \sim_A y$ if there is $a \in A$ such that $a \sqsubseteq x, y$. We may say that $\sim_A$ identifies those elements in R which contain the same total information in their A-part. By R/A we denote the set of equivalence classes of $\sim_A$.

Now assume that $A \to B$ is a dependency where $A \sqsubseteq^\natural B$ and that R is some relation. The result of restricting R to the 'columns' of B is expressed by $\pi_B(R)$. Wherever two elements of $\pi_B(R)$ contain the same total information in their A-part, consistency with $A \to B$ implies that their B-part can be updated to a common (total) value. This amounts to saying that each equivalence class in $\pi_B(R)/A$ has an upper bound in $\mathcal{D}$. Let us denote the resulting set of suprema by $(\pi_B(R)/A)^\sqcup$. Formally we define

Definition. For $A \sqsubseteq^\natural B$ schemes and R a relation in a domain D, we say that R is *weakly consistent* with the dependency $A \to B$ if $(\pi_B(R)/A)^\sqcup$ exists.

Remember the example of physical constants, given in Section 2. Certainly we expect that the name of a constant will imply its value, although the exact numbers will never be known. To say that our database is weakly consistent with the implication *name of constant* $\to$ [*lower bound, upper bound*] amounts to the requirement that the entries for the same constant report intervals with at least one common point.

The reader will have noticed that weak consistency makes no requirement about those elements of the relation R which contain partial information in their A-part. The philosophy here is that any finite set of elements with partial information over some scheme A can be updated in such a way that its elements are pairwise different in their A-part. We may call a domain in which this is always the case *rich* and obtain the following immediate characterization.

Lemma 8 *A domain $\mathcal{D}$ is rich if and only if for each $x \in \mathcal{D}$ the denotation $[\![x]\!]$ of x cannot be covered by a finite set of denotations $[\![y_i]\!]$ with all $y_i \not\sqsubseteq x$.*

With this we can formulate

Theorem 9 *Let $A \sqsubseteq^\natural B$ be schemes in a domain $\mathcal{D}$. Let R be a relation in $\mathcal{D}$. If R is consistent with $A \to B$ then R is weakly consistent with $A \to B$. If $\downarrow A$ is rich and D distributive then the converse also holds. Moreover, if $R' \sqsupseteq^\natural R$ and R' satisfies $A \to B$ then $R' \sqsupseteq^\flat (\pi_B(R)/A)^\sqcup$.*

Proof. Suppose $R' \sqsupseteq^\natural R$ and R' satisfies (A, B), then $\pi_B(R') \sqsupseteq^\flat \pi_B(R)$ and the members of $(\pi_B(R')/A)$ are singleton sets. Thus any class in $(\pi_B(R)/A)$ is bounded above by one of these singletons, and $(\pi_B(R)/A)^\sqcup$ exists. This also establishes the second part of the theorem. Conversely, if $(\pi_B(R)/A)^\sqcup$ exists, we have, for each $a \in A$, the element $b_a = \sqcup\{r \mid r \in \pi_B(R) \text{ and } r \sqsupseteq a\} \in \downarrow B$. Now for each $r \in R \cap \uparrow A$ form the point $r' = b_A \sqcup r$ with $a \in A$ being the unique element of A below r. (This is where the slide condition comes in.) The set R' of these points certainly satisfies $A \to B$. But we also have to update the other elements of R which contain partial information in their A-part. We use the assumption that $\downarrow A$ is rich for this. The set $p_A(R \setminus \uparrow A)$ is a finite poset contained in $\downarrow A$. Because $\downarrow A$ is rich, we can find elements $\bar{r} \in [\![p]\!]_A(r)_A$ such that $r_1 \neq r_2$ implies $\bar{r_1} \neq \bar{r_2}$ and also $\bar{r} \neq p_A(r')$ for all $r' \in R'$. (Given $r \in p_A(R \setminus \uparrow A)$, choose $\bar{r} \in [\![r]\!]_A \setminus (\bigcup\{\uparrow s \mid s \not\sqsubseteq r, s \in p_A(R \setminus \uparrow A)\} \cup \bigcup\{\uparrow r' \mid r' \in R'\})$.) Finally let $\tilde{r}$ be the unique element of B above $\bar{r}$ for each $r \in p_A(R \setminus \uparrow A)$. The set $\tilde{R}$ of all these elements satisfies $A \to B$ and so does $R' \cup \tilde{R}$. $\Box$

Dependencies are often divided [46, 24] into two classes: those like functional dependencies that generate equality constraints, and those that generate new tuples. The "chase" is a procedure that performs all possible inferences on a set R to produce a new set R' where $R' \sqsupseteq^\natural R$. In fact, we can also use functional dependencies in the same way. The co-chain $(\pi_B(R)/A)^\sqcup$ describes the inferences that can be made, given that R is consistent with $A \to B$. In fact the co-chain

$$T = ((\pi_B(R)/A)^\sqcup \bowtie R) \sqcup^\flat R \tag{1}$$

is the least (in $\sqsubseteq^\flat$) set that contains all these inferences. Note that T is the outer join of $(\pi_B(R)/A)^\sqcup$ and R and that $R \sqsubseteq^\natural T$.

6 Universal Relations

Without involving ourselves in a discussion of the usefulness or practicality of the Universal Relation Assumption [47, 48, 21, 6], we now investigate a general characterization of universal relations that shows how the general form of their implementation can be derived from their abstract properties. Behaviorally, a universal relation can be thought of as a simple query language, or transducer, in which the possible queries, or inputs, are sets of column names and the output from the input of a given set of column names is a relation defined on those names. More precisely, we can think of a universal relation as a function $\rho : \mathcal{S}_\mathcal{D} \rightarrow \mathcal{C}_\mathcal{D}$ with the property that $\rho(S)$ is an instance of S, i.e. $\rho(S) \subseteq S$.

In a survey [25] of the various definitions of universal relations Maier *et al* give a condition, "containment", that all reasonable definitions satisfy. The condition, which is also noted in [41], is that if A, B are schemes with $A \sqsubseteq^\natural B$, then $\pi_A(\rho(B)) \subseteq \rho(A)$. This is equivalent to requiring that ρ be monotone as a function from schemes under the natural ordering to the finite cochains $\mathcal{C}_\mathcal{D}$ under the Smyth ordering, i.e. if $A \sqsubseteq^\natural B$ then $\rho(A) \sqsubseteq^\sharp \rho(B)$. There are various ways of obtaining such a function. A particularly simple method is given the *total projection* of an arbitrary subset T of $\mathcal{D}$ onto the schemes of $\mathcal{D}$:

$$\rho(A) = \pi_A(T \cap \uparrow A) \tag{2}$$

(the expression $\pi_S(T \cap \uparrow A)$ is called the total projection of T onto the scheme A.) A more general method is obtained by projecting onto A those subsets T of some collection $\boldsymbol{T}$ of finite subsets of $\mathcal{D}$ that are contained in the upward closure of A:

$$\rho(A) = \cup\{\pi_A(T) \mid T \in \boldsymbol{T} \text{ and } T \subseteq \uparrow A\} \tag{3}$$

Most of the various definitions of universal relations given in [25] appear to be expressible in this form. By using a result that is readily proved from theorem 6,

Lemma 9 *If A is a scheme, and S_1, S_2 are co-chains in $\mathcal{D}$ with $S_1 \sqsupseteq^\sharp A$ and $S_2 \sqsupseteq^\sharp A$ then $\pi_A(S_1 \sqcap^\sharp S_2) = \pi_A(S_1) \cup \pi_A(S_2)$*$\Box$

we can write (3) as $\rho(A) = \pi_A(\sqcap^\sharp\{T \in \boldsymbol{T} \mid T \sqsubseteq^\sharp A\})$. We shall call a universal relation that can be described in this fashion a *closure universal relation* (because this last equation is closely related to the definition of a closure in $(\mathcal{C}_\mathcal{D}, \sqsubseteq^\sharp)$. By taking $\boldsymbol{T}$ as a collection of singleton sets, equation (2) can be seen as a special case of (3). An example of a universal relation satisfying (2) is the *Universal Instance Assumption*, which says that $\rho(A) = \pi_A(I)$ where I is a subset of the maximal elements of $\mathcal{D}$.

Theorem 10 *A universal relation defined by the universal instance assumption is a closure relation.*

The proof follows immediately from the observation that I, being a finite set of maximal elements, is contained in $\uparrow A$ for any scheme A $\Box$

Another reason for believing that closure universal relations are an appropriate class to consider is given by the following result.

Theorem 11 *In the relational domain $\mathcal{D} = \mathcal{L} \to \mathcal{V}_\perp$, any universal relation satisfying the containment condition is a closure universal relation.*

Proof. If ρ is the given universal relation define

$$\rho'(A) = \pi_A(\sqcap^\sharp\{\rho(B) \mid B \in \mathcal{S}_\mathcal{D} \text{ and } \rho(B) \subseteq \uparrow A\}).$$

ρ' is then a closure universal relation, and we need to show that, for any scheme A, $\rho(A) = \rho'(A)$. Because we are dealing with the relational domain, if B is a scheme such that $\phi \neq \rho(B) \subseteq \uparrow A$ then $B \sqsupseteq^\natural A$. Using this fact and the containment condition, whenever $\rho(B) \subseteq \uparrow A$, we must have $\pi_A(\rho(B)) \subseteq \rho(A)$. Hence $\rho(A) = \rho'(A)$ for any scheme A $\Box$

It is not true that any universal relation satisfying the containment condition can be cast in the form of a closure relation. Consider, for example, the domain in figure 6, in which the schemes are $A_1 = \{\perp\}$, $A_2 = \{a_1, a_2, d\}$, $A_3 = \{b_1, b_2, d\}$, $A_4 = \{a_1, a_2, e_1, e_2\}$, $A_5 = \{b_1, b_2, e_1, e_2\}$, and $A_6 = \{c_1, c_2, c_3, c_4, e_1, e_2\}$. Now consider a universal relation ρ such that $\rho(A_1) = \{\perp\}$, $\rho(A_2) = \rho(A_3) = \{d\}$, $\rho(A_4) = \{e_1, e_2\}$, $\rho(A_5) = \{e_1\}$, and $\rho(A_6) = \{e_1\}$, which satisfies the containment condition. If ρ is a closure universal relation, then $\boldsymbol{T}$ (as used in equation 3) must contain a set T which contains e_2 such that $T \subseteq \uparrow A_4$, but T cannot be contained in $\uparrow A_5$ because e_2 is not a member of $\rho(A_5)$. Therefore T must contain a_1 or a_2. But if this happens then $\rho(A_4)$ must also contain a_1 or a_2, which contradicts the definition of ρ.

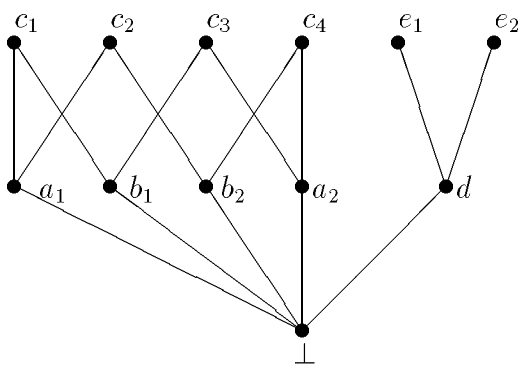

Figure 6: A universal relation not extending to a closure

A more sophisticated example of a universal relation definition arises from the *F-weak instance universal relation* [25]. Suppose we are given a set of schemes $\{R_1, R_2, \ldots, R_n\}$ in $\mathcal{D}$ and instances $r_i \subseteq R_i$, $i \in 1 \ldots n$. Suppose we are also given a set F of functional dependencies and that $\sqcup^\flat\{r_i \mid i \in 1\,..\,n\}$ is consistent with F. Consider the universal relation defined by

$$\rho(C) = \cap\{\pi_C(S') \mid S' \sqsupseteq^\flat r_i, i \in 1 \ldots n, S' \in \mathcal{C}_\mathcal{D}, \text{ and } S' \text{ satisfies } F\} \tag{4}$$

which, for each scheme C defines an instance of C. Let us assume, for simplicity, that $F+$ is generated by the single non-trivial dependency (A, B) where $B \sqsupseteq^\natural A$. From (1) of the previous section, we can write $\rho(C)$ as

$$\rho(C) = \tau_C(((\pi_B(S)/A)^\sqcup \bowtie S) \sqcup^\flat S) \tag{5}$$

where $S = \sqcup^\flat\{r_i \mid i \in 1\,..\,n\}$ and $\tau_C(T)$ is the total projection of a set T onto C, $\tau_C(T) = \pi_C(T \cap \uparrow C)$.

By manipulation of (5) we can now write it in a form consistent with the general form for closure universal relations given in (3). First observe that if S_1, S_2 are co-chains in $\mathcal{D}$, then $\tau_C(S_1 \sqcup^\flat S_2) = \tau_C(S_1) \cup \tau_C(S_2)$. Therefore we can rewrite (5) as

$$\rho(C) = \tau_C((\pi_B(S)/A)^\sqcup \bowtie S) \quad \cup \quad \cup\tau_C(\{r_i \mid R_i \sqsupseteq^\natural C\}) \tag{6}$$

Now consider the set $Q = (\pi_B(S)/A)^{\sqcup}$ which, by the definition of S is $(\pi_B(\sqcup^{\flat}\{r_i \mid i \in 1..n\})/A)^{\sqcup}$, By the distributivity of $(\mathcal{C}_{\mathcal{D}}, \sqsubseteq^{\flat})$, $Q = (\sqcup^{\flat}\{\pi_B(r_i) \mid i \in 1..n\}/A)^{\sqcup}$. A point in Q is the least upper bound of some set of points, each chosen from some $\pi_B(r_i)$ where $R_i \sqsupseteq^{\natural} A$. Let I be the set of indices of all such schemes, $I = \{i \mid i \in 1..n \text{ and } R_i \sqsupseteq^{\natural} A\}$. We can then express Q as

$$Q = \sqcup^{\flat}\{\bowtie_{i \in I'} \pi_B(r_i) \mid I \subseteq I' \subseteq 1..n\} \tag{7}$$

The term $\tau_C((\pi_B(S)/A)^{\sqcup} \bowtie S)$, which is the left-hand component of (6) can therefore be written as the union of total projections of terms of the form

$$r_{i_0} \bowtie \pi_B(r_{i_1}) \bowtie \pi_B(r_{i_2}) \bowtie \cdots \bowtie \pi_B(r_{i_k}) \tag{8}$$

where $R_{i_j} \sqsupseteq^{\natural} A$ for $j \in 1..k$. The right hand component can, trivially, be written in this form too.

We have therefore succeeded in reducing the universal relation definition given in equation (5) to the projection of the union of a set of joins. More importantly, (5) is an example of an "FD-join" expression. A theorem of Maier *et. al.* and Chan [25, 15] shows that the F-weak instance universal relation (5) can be computed as the union of FD-joins. Their proofs work by considering the properties of specific algorithms, whereas by considering the general properties of the spaces involved we have been able to produce a reasonably concise algebraic derivation. It should be noted that the proof outlined here is incomplete. We need to close this off under all functional dependencies; but this presents no difficulties.

7 Higher Order Relations and Other Models

One of the contentions of this paper is that much of our theory of relational databases is independent of the detailed structure of the relational model and depends only on some rather general properties of the spaces out of which we can construct such a model. It should be stressed that we have based the preceding analysis only on the assumption that the underlying space was a domain. Nowhere did we assume that we were dealing with relations, although we frequently appealed to the first-normal-form relations for examples.

In trying to generalize various operations, we had no problem with the natural join, but in order to make projection generalize smoothly when dealing with functional dependencies and universal relations, we had to characterize first independent sets and then schemes. We shall therefore be particularly interested in identifying schemes in these other models. If we can do that, we can be sure that the basic ideas of functional dependencies, universal relations etc., generalize properly.

7.1 Typed first-normal-form relations

We have seen that the domain $\mathcal{L} \to \mathcal{V}_\perp$ of flat records is a domain of the relational model in the preceding sections, and it deserves little extra comment here. As we have noted earlier, this domain is a special case of a product domain.

Given a function F from a set of labels $\mathcal{L}$ to a set of sets $\boldsymbol{S}$, a *labeled product* $\prod_{l \in \mathcal{L}} F(l)$ is the set of functions $f : \mathcal{L} \to \bigcup \boldsymbol{S}$ such that for all $l \in \mathcal{L}$, $f(l) \in F(l)$. If $\boldsymbol{S}$ is a set of domains, then $\prod_{l \in \mathcal{L}} F(l)$ is also a domain, a *domain of labeled products*, under the componentwise ordering, i.e. $f_1 \sqsubseteq f_2$ iff $f_1(l) \sqsubseteq f_2(l)$ for all $l \in \mathcal{L}$. Furthermore, a scheme in a domain of labeled products is a product of schemes, i.e. it is easy to show that:

Lemma 10 *The set of schemes in $\prod_{l \in \mathcal{L}} F(l)$ is the set of labeled products of the form $\prod_{l \in \mathcal{L}} \phi(l)$, where ϕ is any function from $\mathcal{L}$ to $\bigcup \{\mathcal{S}_S \mid S \in \boldsymbol{S}\}$ such that $\phi(l) \in \mathcal{S}_{F(l)}$* $\Box$

Since the domain of flat records $\mathcal{L} \to \mathcal{V}_\perp$ is the domain of labeled product $\prod_{l \in \mathcal{L}} \mathcal{V}_\perp$, where we take $\mathcal{V}_\perp$ as the constant function on $\mathcal{L}$, the above result shows that a scheme in this domain is a product $\prod_{l \in \mathcal{L}} \phi(l)$ where ϕ is any function from $\mathcal{L}$ to $\mathcal{S}_{\mathcal{V}_\perp}$. Since $\mathcal{S}_{\mathcal{V}_\perp} = \{\{\perp\}, \mathcal{V}\}$, each such function $\phi : \mathcal{L} \to \mathcal{S}_{\mathcal{V}_\perp}$ is identified by the subset $L = \{l \mid \phi(l) \neq \{\perp\}\}$ of $\mathcal{L}$ and the corresponding scheme is isomorphic to the set of total functions from L to $\mathcal{V}$. Therefore the set of all schemes in this domain is isomorphic to the set of spaces of total functions $L \to \mathcal{V}, L \subseteq \mathcal{L}$ and is identified by the set of all subsets of $\mathcal{L}$. However, restrictions on these function spaces do not produce schemes, for example

$$\{\{\mathit{Name} \Rightarrow s;\ \mathit{Age} \Rightarrow i; \mathit{Shoe\text{-}size} \Rightarrow i\} \mid s, i \in \mathcal{V}\}$$

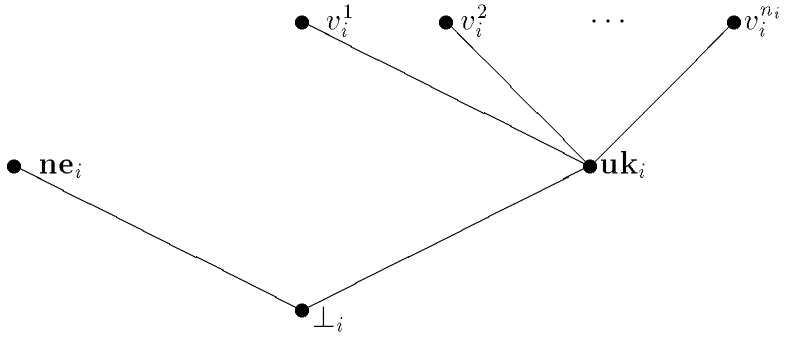

Figure 7: A domain V_i with null values

is not a scheme if $\mathcal{V}$ has more than one element.

If we require that the columns of a relation are "typed", we are given a set of flat domains $\boldsymbol{\mathcal{V}}$ and an assignment of domains in $\boldsymbol{\mathcal{V}}$ to labels in $\mathcal{L}$, i.e. a function $\Phi : \mathcal{L} \to \boldsymbol{\mathcal{V}}$ ($\Phi(l)$ are called "domains" in database parlance). Then the domain of typed flat records $\mathcal{D}_\Phi$ is the domain of labeled products $\mathcal{D}_\Phi = \prod_{l \in \mathcal{L}} \Phi(l)$. A scheme in this domain is a product $\prod_{l \in \mathcal{L}} \phi(l)$ where ϕ is any function $\phi : \mathcal{L} \to \bigcup_{l \in \mathcal{L}} \mathcal{S}_{\Phi(l)}$ such that for all $l \in \mathcal{L}, \phi(l) \in \mathcal{S}_{\Phi(l)}$. Since each $\Phi(l)$ is a flat domain, $\mathcal{S}_{\Phi(l)}$ is either $\{\perp\}$ or the set of all maximal elements in $\Phi(l)$. Thus the set of all schemes in this domain is isomorphic to the set of all product domains of the form $\prod_{l \in L} \Phi(l), L \subseteq \mathcal{L}$. If each $\Phi(l)$ is represented by a type τ_l, then for a finite $L = \{l_1, \ldots, l_n\}$, a scheme $\prod_{l \in L} \Phi(l)$ is represented by the type $\{l_1 : \tau_{l_1}, \ldots, l_n : \tau_{l_n}\}$.

7.2 Null values

Our first "non-flat" example arises from the introduction of null values, which give rise to an ordering on tuples. The framework that we have developed here should allow us to ascribe semantics to the various kinds of null values and to investigate how the mathematical properties generalize.

Combining work in [9, 22, 40] Zaniolo [49] introduced an ordered space V_i with null values shown in figure 7.

$\perp_i$ is interpreted as *no information*; $\mathbf{ne}_i$ means *non-existent*, or *wrong*; $\mathbf{uk}_i$ means *unknown* – a value exists (other than $\mathbf{ne}_i$), but it is not yet known.

Tree-like domains such as this are domains with a particularly simple structure. In fact we can call a domain $\mathcal{D}$ a *tree* if, whenever $x, y \in \mathcal{D}$ and $x \sqcup y$ exists then $x \sqsubseteq y$ or $y \sqsubseteq x$. A *section* of a tree $\mathcal{D}$ is a set S such that any path in $\mathcal{D}$ from the root $(\perp)$ to a leaf contains exactly one member of S. The following results characterize independent sets and schemes in a tree.

Lemma 11 *$\mathcal{D}$ is a tree iff $\mathcal{C}_\mathcal{D} = \mathcal{I}_\mathcal{D}$ (i.e. the co-chains are the independent sets)*$\Box$

Lemma 12 *S is a section of $\mathcal{D}$ iff it is a scheme for $\mathcal{D}$*$\Box$

For example, the schemes for V_i in figure 7 are $\{\perp_i\}$, $\{\mathbf{ne}_i, \mathbf{uk}_i\}$ and $\{\mathbf{ne}_i, v_i^1, \ldots, v_i^{n_i}\}$.

We can use this to define domains of typed records with null values by simply replacing flat domains with tree-like domains in the previous development. Given a set $\boldsymbol{\mathcal{T}}$ of tree-like domains and a type assignment $\Phi : \mathcal{L} \rightarrow \boldsymbol{\mathcal{T}}$, a domain of typed records $\mathcal{D}_\Phi$ is the domain $\mathcal{D}_\Phi = \prod_{l \in \mathcal{L}} \Phi(l)$ of labeled products. A scheme in this domain is a product $\prod_{l \in \mathcal{L}} \phi(l)$ where ϕ is a function $\phi : \mathcal{L} \rightarrow \bigcup_{l \in \mathcal{L}} \mathcal{S}_{\Phi(l)}$ such that for all $l \in \mathcal{L}, \phi(l) \in \mathcal{S}_{\Phi(l)}$. Unlike the case of typed flat records, $\mathcal{S}_{\Phi(l)}$ may contain schemes which are neither $\{\perp\}$ nor the set of maximal elements in $\Phi(l)$ and the set of schemes in this domain is no longer isomorphic to the set of products of the form $\prod_{l \in L} \Phi(l), L \in \mathcal{L}$. In order to represent schemes in this domain in a type system, we need to define "scheme-types" to represent schemes $\mathcal{S}_\mathcal{T}, \mathcal{T} \in \boldsymbol{\mathcal{T}}$. We will show an example of such definition in the next section.

This allows us to establish that the whole apparatus of functional dependencies, universal relations, etc. works smoothly in the domain of relations with null values, i.e. relations defined over tree-like domains.

To take an example, in a payroll database, the values $\{v_i^1, \ldots, v_i^{n_i}\}$ could be the state tax rate with $\mathbf{ne}_i$ being used when such a tax was inappropriate, e.g.,. for overseas employees. There is then a functional dependency ADDRESS$\rightarrow$ $\{\mathbf{ne}_i, v_i^1, \ldots, v_i^{n_i}\}$ and an inferred dependency ADDRESS$\rightarrow$ $\{\mathbf{ne}_i, \mathbf{uk}_i\}$. The investigation of such dependencies may be useful when attempting to do database design on databases with exceptional values such as those investigated in [10].

7.3 Record structures

In programming languages such as Pascal, record types are constructed both by giving a labelled set of fields and by giving a **case** statement or discriminated union. Moreover record types can be components of other record types, and we can carry this construction to any depth. The domains of such records allows us a further generalization of the domains we have just considered. These domains can be also regarded as the domain of feature structures which are used to represent linguistic information [43].

In the previous sections, we have constructed domains and their schemes of first-normal form relations with null values by using labeled product constructors. By simply iterating this construction process, we can construct domains and schemes of general record structures without discriminated union. Domains corresponding to discriminated union can be constructed by labeled sum constructors.

Given a function F from a set of labels $\mathcal{L}$ to a set of sets $\boldsymbol{S}$, a *labeled sum* $\sum_{l\in\mathcal{L}} F(l)$ is the set of pairs $\{< l, v >\mid v \in F(l)\}$. If $\boldsymbol{S}$ is a set of domains, we define the *domain of labeled sums* $\sum^{\perp}_{l\in L} F(l)$ to be the set $\{< l, v >\mid v \in F(l)\} \cup \{\perp\}$. This is indeed a domain under the ordering defined as $x \sqsubseteq y$ if and only if either $x = \perp$ or $x =< l, v >$ and $y =< l, v'$ and $v \sqsubseteq v'$. Corresponding to the result for labelled products (lemma 10), a scheme in a domain of labeled sums is a labeled sum of schemes, i.e. it is easy to show that:

Lemma 13 *A scheme in* $\sum^{\perp}_{l\in\mathcal{L}} F(l)$ *is either the singleton set* $\{\perp\}$ *or a labeled sum* $\sum_{l\in\mathcal{L}} S(l)$*, where* S *is any function from* $\mathcal{L}$ *to* $\bigcup\{\mathcal{S}_S \mid S \in \boldsymbol{S}\}$ *such that* $S(l) \in \mathcal{S}_{F(l)}\Box$.

Starting with given primitive domains such as the flat domain of integers, we can now construct domains of record structures by applying product and sum constructions. We can then identify the set of schemes in those domains. Suppose we are given primitive domains $\mathcal{B}_1, \ldots, \mathcal{B}_n$ with corresponding sets of schemes $\mathcal{S}_{\mathcal{B}_1}, \ldots, \mathcal{S}_{\mathcal{B}_n}$. Then we can define the family $\boldsymbol{Dom}$ of domains with associated sets of schemes generated by $\mathcal{B}_i$'s as:

(1) $\mathcal{B}_i \in \boldsymbol{Dom}$. The associated set of scheme is $\mathcal{S}_{\mathcal{B}_i}$,

(2) If $\boldsymbol{\mathcal{D}} \subseteq \boldsymbol{Dom}$ with associated sets of schemes $\mathcal{S}_{\mathcal{D}}, \mathcal{D} \in \boldsymbol{\mathcal{D}}$, then for any function $\Phi : \mathcal{L} \to \boldsymbol{\mathcal{D}}$, $\mathcal{D}_{\Phi} = \prod_{l \in \mathcal{L}} \Phi(l) \in \boldsymbol{Dom}$ with the set of schemes $\mathcal{S}_{\mathcal{D}_{\Phi}} = \{\prod_{l \in \mathcal{L}} \phi(l) \mid \exists \phi : \mathcal{L} \to \bigcup_{\mathcal{D} \in \boldsymbol{\mathcal{D}}} \mathcal{S}_{\mathcal{D}}.\ \forall l \in \mathcal{L}.\ \phi(l) \in \mathcal{S}_{\Phi(l)}\}$.

(3) If $\boldsymbol{\mathcal{D}} \subseteq \boldsymbol{Dom}$ with corresponding sets of schemes $\mathcal{S}_{\mathcal{D}}, \mathcal{D} \in \boldsymbol{\mathcal{D}}$, then for any function $\Theta : \mathcal{L} \to \boldsymbol{\mathcal{D}}$, $\mathcal{D}_{\Theta} = \sum^{\perp}_{l \in \mathcal{L}} \Theta(l) \in \boldsymbol{Dom}$ with the set of schemes $\mathcal{S}_{\mathcal{D}_{\Theta}} = \{\sum_{l \in \mathcal{L}} \theta(l) \mid \exists \theta : \mathcal{L} \to \bigcup_{\mathcal{D} \in \boldsymbol{\mathcal{D}}} \mathcal{S}_{\mathcal{D}}.\ \forall l \in \mathcal{L}.\ \theta(l) \in \mathcal{S}_{\Theta(l)}\}$.

$\boldsymbol{Dom}$ corresponds to domains of record structures generated from primitive values in $\mathcal{B}_1, \ldots, \mathcal{B}_n$.

We give an example of concrete representation of domains of record structures. By the analogy of a type system of a porgramming language, we call expression representing domains *types* and define the membership relation between records and domains as typing rules. We will comment more on the relationship between domains and types in a programming language later. We start with types. A *type expression* is one that can be constructed by the following rules:

(1) $B_1, \ldots, B_n$, the (names of) base types such as *int, bool, string* etc. are type expressions.

(2) If $\tau_1, \tau_2, \ldots, \tau_n$ are type expressions then $\{l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n\}$ is a type expression.

(3) If $\tau_1, \tau_2, \ldots, \tau_n$ are type expressions then $[l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n]$ is a type expression.

The notation $[l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n]$ indicates a discriminated union. An example of such a type expression is

$$\begin{aligned}\tau_1 = \{&Name{:}string;\ Age{:}int;\ Status{:}[Employee{:}\{Office{:}string;\ Extension{:}int\};\\ &\qquad\qquad\qquad\qquad Consultant{:}\{Address{:}string;\ Telephone{:}int\}]\}\end{aligned}$$

The syntax for records is similarly defined:

(1) For each base type B, we assume that we are given the corresponding primitive domain $\mathcal{B}$ such as the flat domain $N_{\perp}$ of integers. Then elements in $\mathcal{B}$ are records. $\perp_{\mathcal{B}}$ represents a null value in $\mathcal{B}$.

(2) If $r_1, r_2, \ldots, r_n$ are records then $\{l_1 \Rightarrow r_1; l_2 \Rightarrow r_2; \ldots; l_n \Rightarrow r_n\}$ is a record.

(3) If r is a record, $[l \Rightarrow r]$ is a record.

(4) If τ is a discriminated union type then $\perp_\tau$ is a record.

The following is an example of record:

$$r_1 = \{\mathit{Name}{\Rightarrow}'J.\ Doe';\ \mathit{Age}{\Rightarrow}21;\ \mathit{Status}{\Rightarrow}[\mathit{Employee}{\Rightarrow}\{\mathit{Office}{\Rightarrow}G7;\ \mathit{Extension}{\Rightarrow}5556\}]\}$$

Moreover, we regard the record r_1 having the type τ_1. Formally, a record r has type τ if one of the following conditions hold:

(1) $r \in \mathcal{B}$ and τ is the base type B corresponding to $\mathcal{B}$.

(2) $r = \{l_1 \Rightarrow r_1; l_2 \Rightarrow r_2; \ldots; l_n \Rightarrow r_n\}$, $\tau = \{l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n\}$, and r_i has type τ_i for $1 \leq i \leq n$.

(3) $r = [l_i \Rightarrow r_i]$, $\tau = [l_1 : \tau_1; l_2 : \tau_2; \ldots; l_m : \tau_m]$, $i \leq m$, and r_i has type τ_i.

(4) $r = \perp_\tau, \tau = [l_1 : \tau_1, \ldots, l_n : \tau_n]$

Records are ordered by the following rules:

(1) $v \sqsubseteq v'$ if $v, v' \in \mathcal{B}$ and v, v' are ordered in $\mathcal{B}$.

(2) $\{l_1 \Rightarrow r_1; l_2 \Rightarrow r_2; \ldots; l_n \Rightarrow r_n\} \sqsubseteq \{l_1 \Rightarrow r'_1; l_2 \Rightarrow r'_2; \ldots; l_n \Rightarrow r'_n\}$ if $r_i \sqsubseteq r'_i$ for all $1 \leq i \leq n$.

(3) $[l \Rightarrow r] \sqsubseteq [l \Rightarrow r']$ if $r \sqsubseteq r'$.

(4) $\perp_\tau \sqsubseteq \perp_\tau$ for any discriminated union type τ.

(5) $\perp_{[l_1:\tau_1,\ldots,l_n:\tau_n]} \sqsubseteq [l_i \Rightarrow r]$ if $1 \leq i \leq n$ and r has the type τ_i.

Informally, one record is better than another if it has better values in same fields. For example, if

$$r_2 = \{\mathit{Name}{\Rightarrow}'J.\ Doe';\ \mathit{Age}{\Rightarrow}\perp_{int};\ \mathit{Status}{\Rightarrow}[\mathit{Employee}{\Rightarrow}\{\mathit{Office}{\Rightarrow}G7;\ \mathit{Extension}{\Rightarrow}\perp_{int}]\}$$

then $r_2 \sqsubseteq r_1$. From these definitions we can immediately see that the set of all records of a type τ is a domain $\mathcal{D}_\tau$ belonging to the family of domains $\boldsymbol{Dom}$ constructed from the set of primitive domains $\mathcal{B}_1, \ldots, \mathcal{B}_n$ and the ordering relation on records corresponds exactly to the orderings on domains in $\boldsymbol{Dom}$.

We next deine the syntax of *scheme-types* for a type τ. τ' is a scheme-type for τ if:

(1) τ is a base type and $\tau' = \tau$ or $\tau' = unit_\tau$. $unit_\tau$ denotes the trivial scheme $\{\perp_\mathcal{B}\}$ in $\mathcal{B}$.

(2) $\tau' = \{l_1 : \tau'_1; l_2 : \tau'_2; \ldots; l_n : \tau'_n\}$, $\tau = \{l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n\}$ and τ'_i is a scheme-type for τ_i, for $1 \leq i \leq n$.

(3) $\tau' = [l_1 : \tau'_1; l_2 : \tau'_2; \ldots; l_n : \tau'_n]$, $\tau = [l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n]$, and τ'_i is a scheme-type for τ_i, for $1 \leq i \leq n$.

(4) $\tau' = unit_\tau$ and τ is any discriminated union type. $unit_\tau$ denotes the trivial scheme $\{\perp_\tau\}$ in $\mathcal{D}_\tau$.

The following is a scheme-type of the type τ_1 defined in our example of a record type above:

$$\begin{aligned}\tau_2 = \{&Name{:}string;\ Status{:}[Employee{:}\{Office{:}string;\ Extension{:}unit_{int}\};\\ &\qquad\qquad\qquad\qquad Consultant{:}\{Address{:}string;\ Telephone{:}unit_{int}\}]\}\end{aligned}$$

Moreover, we regard the record r_2 having the above scheme-type τ_2. Formally, a record r has a scheme type τ if:

(1) $r \in \mathcal{B}_i$ and $\tau = B_i$.

(2) $r = \perp_{\mathcal{B}_i}$, $\tau = unit_{B_i}$.

(3) $r = \{l_1 \Rightarrow r_1; \ldots, l_n \Rightarrow r_n\}$, $\tau = \{l_1 : \tau_1, \ldots, l_n : \tau_n\}$ and r_i has the scheme-type τ_i for $1 \leq i \leq n$.

(4) $r = [l_i \Rightarrow r_i]$,$\tau = [l_1 : \tau_1, \ldots, l_n : \tau_n]$, $i \leq n$ and r_i has scheme-type τ_i.

A	B	
	C	D
a_1	c_1	d_1
	c_2	d_2
a_2	c_4	d_1
	c_3	d_3
	c_2	d_1

(a)

A	B	
	C	D
a_1	c_1	d_1
a_1	c_2	d_2
a_2	c_4	d_1
a_2	c_3	d_3
a_2	c_2	d_1

(b)

Figure 8: Restricted higher-order relation and equivalent relation

(5) $\bot_{\tau'}$ and τ is any scheme-type of any discriminated union type τ'.

Then by the definition of the scheme-types, we can also see that the set of all records of the scheme-type τ' for the type τ is a scheme in $\mathcal{D}_\tau$.

Sets of records belonging to a given type therefore form an interesting generalization of first-normal-form relations for which we can define relational operations, functional dependencies etc.

7.4 Structures that contain sets

An extension to the relational model that has recently enjoyed some popularity is the study of higher-order relations [17, 1, 35, 32]. In these model a value in a tuple can itself be a set of values, i.e. another relation. In order to obtain a class of higher-order relations that behave well under relational operations, [35] describes *partition normal form* relations. In such relations the attributes with simple (atomic) values functionally determine the attributes with higher-order values, which must also be in partition normal form. However, because of this severe restriction, sets are not treated as first-class values in this model. Indeed, it is not hard to show that partition-normal form relations are isomorphic to relations over record structures (without labeled sums) defined in the previous section. For example, the relation (a) in figure 8 is equivalent to the relation (b).

In order to obtain a data model in which sets are treated as first-class values, we need to construct a space of sets as a domain. Since, in defining various database operations, we have only assumed that the underlying space is a domain, once we have done this then sets can be also treated as regular values. In order to construct a domain of sets, we need to define an ordering on sets as database values. One obvious possibility is to treat the space of sets as a flat domain so that two sets are comparable iff there are equal. However, as we have seen, a flat domain has only two schemes, the set of all maximal elements and the trivial scheme containing only bottom element, and does not yield interesting structures.

A second possibility is to regard sets as ordered by $\sqsubseteq^\flat$, which is what Banchilhon used in his complex object model [7]. Given a domain $\mathcal{D}$, it can be shown [42] that we can construct a domain $\mathcal{P}^\flat(\mathcal{D})$ corresponding to the space of sets of elements in $\mathcal{D}$ ordered by $\sqsubseteq^\flat$ (the Hoare powerdomain of $\mathcal{D}$). Since $\mathcal{P}^\flat(\mathcal{D})$ is a domain, the results of previous sections are readily applicable. However, it is probably rather difficult to find semantics of a natural join since a natural join is determined by the ordering $\sqsubseteq^\sharp$ and therefore database sets and sets appears as values in database objects are treated differently. We should also note that, since $\mathcal{P}^\flat(\mathcal{D})$ is a lattice,

Lemma 14 *For a domain $\mathcal{D}$, the schemes in $\mathcal{P}^\flat(\mathcal{D})$ are the singleton sets $\{\{d\}\}$ where $d \in \mathcal{D}$* $\Box$

which means that functional dependencies in such a domian are rather trivial constraints.

Another possibility is to consider sets as values ordered by $\sqsubseteq^\sharp$, which is done in [12, 13, 29]. Smyth showed that [45] for any domain $\mathcal{D}$, a domain $\mathcal{P}^\sharp(\mathcal{D})$ corresponding to the space of of sets of elements in $\mathcal{D}$ ordered by $\sqsubseteq^\sharp$, called Smyth powerdomain of $\mathcal{D}$, can be constructed. Under this approach, a natural join can be given coherent semantics. Again there are no non-trivial schemes in $\mathcal{P}^\sharp(\mathcal{D})$. However, if we relax our definition of a scheme, we can make some progress. Recall that a scheme A is an independent set in a domain $\mathcal{D}$ satisfying

$$p_A(\mathcal{D}) = \underline{A}$$

and

$$\forall x \in \mathcal{D}. p_A^*[\![x]\!]_{\mathcal{D}} = [\![p_A(x)]\!]_{\underline{A}}$$

One way to generalize this is to specify directly a subset of $\mathcal{D}$ that is not necessarily downward closed. We say that a subset S of $\mathcal{D}$ is a *generalized scheme* in $\mathcal{D}$ if (1) S is closed under bounded join, (2) S has a minimal element and (3) the set of maximal elements $maxset(S)$ of S satisfies the second condition of schemes, i.e. $\forall x \in \mathcal{D}.p^*_S[\![x]\!]_{\mathcal{D}} = [\![p_S(x)]\!]_S$ where $p_S(x) = \sqcup\{s \mid s \in S, s \sqsubseteq x\}$. The original definition of schemes is a special case of generalized schemes. We can then find interesting schemes in $\mathcal{P}^\sharp(\mathcal{D})$.

Lemma 15 *If S is a generalized scheme in a domain $\mathcal{D}$ then the set $\mathcal{P}^\sharp(S)$ is a generalized scheme in $\mathcal{P}^\sharp(\mathcal{D})$*

This suggests that if we regard sets as values ordered by $\sqsubseteq^\sharp$, then the previously described type systems can be extended to include a set type constructor by adding the following rules:

(1) If τ is a type then $\{\tau\}$ is a type.

(2) If τ' is a scheme type of τ then $\{\tau'\}$ is a scheme-type of $\{\tau\}$.

(3) If $v_1, \ldots, v_n$ are database objects of type τ then $minset(\{v_1, \ldots, v_n\})$ is a database object of type $\{\tau\}$.

In the third rule, a given set of database objects is coerced to a canonical representative of an element in $\mathcal{P}^\sharp(\mathcal{D})$ by taking its minimal elements. Natural join and projection work properly on the extended structures. Figure 9 shows an example of a natural join in the domain of records extended by these rules.

One restriction of the above approach is that we presuppose the meaning of sets of database objects by chosing the ordering $\sqsubseteq^\sharp$, i.e. sets are overdescribing some desired set of objects. This choice may not be appropriate for some applications. An idea that merits further investigation is to look at partial descriptions that consist of pairs of sets: a complete and a consistent description of some target set. This may be particularly valuable in constructing a semantics for database merging [27] where the individual databases may not form a complete description of the real world.

$$
\begin{aligned}
r_1 \;=\; \{&\{Pname \Rightarrow 'Nut'; Supplier \Rightarrow \{\ \{Sname \Rightarrow 'Smith'; City \Rightarrow 'London'\};\\
&\qquad\qquad\qquad\qquad\qquad\qquad \{Sname \Rightarrow 'Blake'; City \Rightarrow 'Paris'\}\}\};\\
&\{Pname \Rightarrow 'Bolt'; Supplier \Rightarrow \{\{Sname \Rightarrow 'Blake'; City \Rightarrow 'Paris'\};\\
&\qquad\qquad\qquad\qquad\qquad\qquad \{Sname \Rightarrow 'Adams'; City \Rightarrow 'Athens'\}\}\}\}
\end{aligned}
$$

$$
\begin{aligned}
r_2 = \{&\{Pname \Rightarrow 'Nut'; Supplier \Rightarrow \{\{City \Rightarrow 'Paris'\}\}; Qty \Rightarrow 100\};\\
&\{Pname \Rightarrow 'Bolt'; Supplier \Rightarrow \{\{City \Rightarrow 'Paris'\}\}; Qty \Rightarrow 200\}\}
\end{aligned}
$$

$$
\begin{aligned}
r_1 \bowtie r_2 = \{&\{Pname \Rightarrow 'Nut'; Supplier \Rightarrow \{\{Sname \Rightarrow 'Blake'; City \Rightarrow 'Paris'\}\};\\
&\ Qty \Rightarrow 100\};\\
&\{Pname \Rightarrow 'Bolt'; Supplier \Rightarrow \{\{Sname \Rightarrow 'Blake'; City \Rightarrow 'Paris'\}\};\\
&\ Qty \Rightarrow 200\}\}
\end{aligned}
$$

Figure 9: Natural Join of Higher-order Relations

7.5 Recursive structures

It is reasonable to suppose that we can also generalize database theory to work for recursive types, which can be used to give a type to unbounded structures such as lists. For example, given a domain $\mathcal{D}$ represented by a type τ, we can define a type for τ-lists as the type satisfying the folloiwng equasion:

$$list(\tau) = [null : \{\}; nonnull : \{first : \tau; rest : list(\tau)\}]$$

This is the type of all lists of elements in $\mathcal{D}$. Then for any scheme-type τ' for τ, $list(\tau')$ is a scheme-type for $list(\tau)$. There are also other scheme-type for $list(\tau)$ than in the above form. For example, the following is also a scheme-type for $list(\tau)$ that corresponds to the set of all lists of length less than or equal to one.

$$onelist = [null : \{\}; nonnull : \{first : \tau; rest : unit_{list(\tau)}\}]$$

where $unit_{list(\tau)}$ is the scheme-type $list(unit_\tau)$ for $list(\tau)$.

The domain $\mathcal{D}_{list(\tau)}$ corresponding to $list(\tau)$ can be defined as the domain equation:

$$\mathcal{D}_{list(\tau)} = Null + (\mathcal{D} \times \mathcal{D}_{list(\tau)})$$

where $Null$ is the trivial one element domain. Let S be the scheme in $\mathcal{D}$ corresponding to the scheme-type τ'. Then the scheme corresponding to the scheme-type $list(\tau')$ is the set of maximal elements in the domain defined by the equation:

$$\mathcal{D}_{list(\tau')} = Null + (\underline{S} \times \mathcal{D}_{list(\tau')})$$

The scheme corresponding to the scheme-type *onelist* can be also defined.

The general form of schemes in recursive types such as these requires further investigation.

8 Conclusion and Further Investigation

We have tried to show that the application of domain theory allows us to provide a clean semantics for relational databases and provides a generalization of many of the ideas in relational database theory – especially those concerned with database design – into a large class of higher-order and recursive structures.

One major limitation of our work is that our characterization of the relational databases is restricted to a single domain. Operations and notions such as join and functional dependency are defined only within a given domain. It is however desirable to allow databases to contain values of different domains. This becomes essential if we want to treat values in a database as typed data structures and to integrate them into a type system of a programming language. In previous section we have constructed a collection of domains of records. As we suggested, each domain corresponds to a type in a type system of a programming language. In such a type system, it is natural to represent a database as a collection of relations of different types. Our formalism cannot be directly applied to such a database. One way to overcome this limitation would be to develop a theory of the relationship between various domains and to extend our characterization of the relational databases to a family of domains. [29] proposed one such theory for join and projection and showed that a family of database domains can be integrated in an ML style type system. In [31] we have also shown that ML type inference method can be generalized to such an integrated type system. We further hope that the theory of functional dependencies and universal relations we have developed in this paper can be also generalize to families of domains.

Finally we should note that in database programming languages [8, 3, 44], in knowledge bases [11] and in Aït-Kaci's [2] calculus for type subsumption the ordering is not completely derived from the structure of the objects themselves. There is also an imposed lattice or partial order of "entities", "concepts", or "head-terms". The possibility of generalizing relational database notions into these systems may require these imposed orderings to have certain properties.

Acknowledgements

We gratefully acknowledge the assistance of Aaron Watters and David MacQueen in the development of these ideas. The referees made numerous useful comments that helped to improve the presentation.

References

[1] S. Abiteboul and N. Bidoit. Non First Normal Form Relations to Represent Hierarchically Organized Data. In *Proc. 3rd ACM PODS*, Waterloo, Ontario, Canada, 1984.

[2] H. Aït-Kaci. An Algebraic Semantics Approach to the Effective Resolution on Type Equations. *Theoretical Computer Science*, 45:293–351, 1986.

[3] A. Albano, L. Cardelli, and R. Orsini. Galileo: A Strongly Typed, Interactive Conceptual Language. *ACM Transactions on Database Systems*, 10(2):230–260, 1985.

[4] M. P. Atkinson. Programming languages and databases. In S. B. Yao, editor, *The fourth international conference on Very Large Data Bases, Berlin, West Germany*, pages 408–419, IEEE & ACM, September 1978.

[5] M.P. Atkinson and O.P. Buneman. Types and Persistence in Database Programming Languages. *ACM Computing Surveys*, June 1987.

[6] P. Atzeni and D. S. Parker. Assumptions in relational database theory. In *Proc. 1st ACM Symp on Principles of Database Systems*, pages 1–9, 1982.

[7] F. Bancilhon and S. Khoshafin. A Calculus for Complex Objects. In *Proc. ACM Conference on Principles of Database Systems*, 1986.

[8] P. Bernstein, J. Mylopoulos, and H.K.T. Wong. A Language Facility for Designing Database Intensive Applications. *ACM Transactions on Database Systems*, 5(2), June 1980.

[9] J. Biskup. A Formal Approach to Null Values in Database Relations. In *Advances in Data Base Theory Vol 1*, Prenum Press, New York, 1981.

[10] A. Borgida. Thoughts on accomodating exceptions to (type) constraints in Information Systems. In *Proceedings of the Appin workshop on Persistence and Data Types Persistent Programming Research Report 16*, pages 275–280, August 1985.

[11] R.J. Brachman and J.G. Schmolze. An Overview of the KL-one Knowledge Representation System. *Cognitive Science*, 9(3), 1985.

[12] O.P. Buneman. Data Types for Data Base Programming. In *Proceedings of the Appin workshop on Persistence and Data Types Persistent Programming Research Report 16*, Computer Science Department, University of Glasgow, 1985.

[13] P. Buneman and A. Ohori. A Domain Theoretic Approach to Higher-order Relations. In *International Conference on Database Theory, Lecture Notes in Computer Science 243*, Springer-Verlag, 1986.

[14] J. Cartmell. Formalising the Network and Hierarchical Data Models – an Application of Categorical Logic. In *Category Theory and Computer Programming, LNCS 240*, pages 466–492, Springer-Verlag, September 1985.

[15] E.P.F. Chan. Optimal Computation of Total Projections with Unions of Chase Join Expressions. In *Proceedings of ACM SIGMOD Conference*, pages 149–163, 1984.

[16] S.S. Cosmdakis, P.C. Kanellakis, and N Spyratos. *Partition Semantics for Relations*. Technical Report, Brown University, December 1985.

[17] P.C. Fischer and S.J. Thomas. Operators for Non-First-Normal-Form Relations. In *Proc. IEEE COMPSAC*, 1983.

[18] N. Hammer and D. McLeod. Database Description with SMD: A Semantic Database Model. *ACM Transactions on Database Systems*, 6(3):351–386, September 1981.

[19] R. Hull and C.K. Yap. The Format Model: a theory of database organization. *Journal of the ACM*, 31(3):518–537, 1984.

[20] H. Jurgensen and D. Simovici. Towards an Abstract Theory of Dependency Constraints. *Information Systems*, 41, 1987.

[21] W. Kent. Consequences of assuming a universal relation. *ACM Transactions on Database Systems*, 6:331–347, 1983.

[22] Y. Lien. On the Equivalence of Database Models. *Journal of the ACM*, 29(2):333–362, April 1982.

[23] W. Lipski. On Semantic Issues Connected with Incomplete Information Databases. *ACM Transactions on Database Systems*, 4(3):262–296, September 1979.

[24] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[25] D. Maier, D. Rozenshtein, and D.S. Warren. Window Functions. In P. Kanellakis and F.P. Preparata, editors, *Advances in Computing Research Volume 3, The Theory of Databases*, pages 213–246, JAI Press Inc., London, England, 1986.

[26] T.H. Merrett. *Relational Information Systems*. Reston Publishing Co, Prentice-Hall, Reston, Virginia, USA, 1984.

[27] A. Motro and P. Buneman. Constructing Superviews. In *SIGMOD*, 1981.

[28] A. Ohori. *Denotational Semantics of Relational Databases*. Master's thesis, Department of Computer and Information Science, University of Pennsylvania, 1986.

[29] A. Ohori. Semantics of Types for Database Objects. In *Proc. Internat. Conf. on Database Theory*, Lecture Notes in Computer Science, Vol. 326 (Springer, Berlin, 1988), pages 239–251; Extended version submitted to a special issue of *Theoret. Comput. Sci.*.

[30] A. Ohori. Orderings and Types in Databases. In *Workshop on Database Programming Languages*, pages 149–163, Roscoff, France, September 1987.

[31] A. Ohori and P. Buneman. Type Inference in a Database Programming Language. In *Proc. ACM Conference on LISP and Functional Programming*, Snowbird, Utah, July 1988.

[32] Z. Özsoyoğlu and L. Yuan. A Normal Form for Nested Relations. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 251–260, Portland, March 1985.

[33] G. D. Plotkin. A Powerdomain Construction. *Siam Journal Comput.*, 5, September 1976.

[34] N. Rishe. On Denotational Semantics of Data Bases. In *Mathematical Foundations of Programming Semantics, Lecture Notes in Computer Science 239*, pages 249–274, Springer-Verlag, April 1985.

[35] A.M. Roth, H.F. Korth, and A. Silberschatz. *Extended Algebra and Calculus for ¬1NF Relational Databases.* Technical Report TR-84-36, Department of Computer Sciences, The University of Texas at Austin, 1984. revised 1985.

[36] M.A. Roth, H.F. Korth, and A. Silberschatz. *Null Values in ¬1NF Relational Databases.* Technical Report TR-85-32, Department of Computer Sciences, The University of Texas at Austin, December 1985.

[37] W. C. Rounds and R. Kasper. *A Complete Logical Culculus for Record Structures Representing Linguistic Information.* Technical Report, Electrical Engineering and Computer Department, University of Michigan, Ann Arbor, Michigan 48109, 1985.

[38] D.A. Schmidt. *Denotational Semantics, A Methodology for Language Development.* Allyn and Bacon, 1986.

[39] J.W. Schmidt. Some High Level Language Constructs for Data of Type Relation. *ACM Transactions on Database Systems*, 5(2), 1977.

[40] E. Sciore. *Null Values, Updates, and Incomplete Information.* Technical Report, Department of Electrical Engineering and Computer Science, Princeton University, 1979.

[41] E. Sciore. *The universal instance and database design.* PhD thesis, Princeton University, 1980.

[42] D. Scott. Domains for Denotational Semantics. In *ICALP*, July 1982.

[43] S.M. Shieber. An Introduction to Unification-Based Approaches to Grammar. In *Proc. 23rd Annual Meeting of the Association for Computational Linguistics*, 1985.

[44] J. M. Smith, S. Fox, and T. Landers. *ADAPLEX: Rationale and Reference Manual.* Computer Corpolation of America, Four Cambridge Center, Cambridge, Massachusetts 02142, second edition, 1983.

[45] M.B. Smyth. Power Domains. *Journal of Computer and System Sciences*, 16(1):23–36, 1978.

[46] J.D. Ullman. *Principle of Database Systems.* Pittman, second edition, 1982.

[47] J.D. Ullman. The U.R. strikes back. In *Proc 1st ACM Symp on Principles of Database Systems*, pages 10–22, 1982.

[48] J.D. Ullman. Universal Relation interfaces for database systems. In *Proc IFIP*, September 1983.

[49] C. Zaniolo. Database Relation with Null Values. *Journal of Computer and System Sciences*, 28(1):142–166, 1984.