A Relational Algebra for Complex Objects Based on Partial Information*

Leonid Libkin†

Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA 19104, USA

Abstract

We study an approach to relational databases which treats relations not as subsets of a Cartesian product but as subsets of some domain – a partially ordered space of descriptions. This approach permits generalizations of relations that admit null values and variants. In previous work it was shown how to define the notion of a relation *scheme* in such a setting. Here we provide a characterization of a scheme that is more general, and show that operations analogous to projection, selection and join retain the desired properties. Schemes also allow us to develop dependency theory for such generalized relations. An extension of this model is described which admits a set constructor and is therefore useful for the study of higher-order relations and their generalizations.

1 Introduction

It has recently been discovered [5] that a representation of the underlying principles of relational database theory can be found in the theory of domains which is the basis of the denotational semantics of programming languages. This representation does not take into account the details of the data structure and, therefore, allows us to extend the main principles of relational databases to much more general constructions. Use of domain theory in the generalization of relational databases may also help to establish the connection between data models and types, i.e. to represent database objects (not necessarily relational databases) as typed objects in programming languages.

There have been made a number of attempts to generalize relational databases giving up the first-normal-form assumption, see [1, 4, 6, 9, 10, 13, 16, 17, 18, 20]. They can be divided into two categories. The first one consists of models that do not contain sets. Usually it means that they admit null-values and/or record structures, the latter including also **case**, or discriminated union. In this case [5] provides us with the idea how to develop the relational theory. We will discuss in details the main definition of *scheme* which was used to introduce projections. The definition of scheme along with the idea to represent database objects as antichains in domains is the main tool to generalize relational databases that was used in [5]. But the definition of scheme in that paper was chosen rather arbitrarily in order to make certain properties of the first-normal-form relational databases generalize smoothly.

Another definition of scheme will be introduced which seems to be more natural and then it will be proved that the main results of [5] remain correct if we substitute the definition of scheme. We will also show that the definition of [5] assumes that a database is like-first-normal-form, that is, the domains in which the two definitions coincide,

*This paper was published in the Proceedings of the 3rd Symposium on Mathematical Fundamentals of Database and Knowledge Base Systems, Rostock, Germany, May 1991, *Springer Lecture Notes in Computer Science*, **495** (1991), 29-43.

†Supported in part by NSF Grants IRI-86-10617 and CCR-90-57570 and ONR Grant NOOO14-88-K0634.

behave like domains of relations that do not admit relation-valued attributes.

For the structures that do not contain sets we will discuss the concept of the *complement* of a scheme. This concept is necessary in order to introduce multivalued dependencies. Multivalued dependencies having been introduced, we may try to define *join* to generalize the result that establishes connection between joins and multivalued dependencies [24, 16]. The concept of join for the domain model was introduced in [5] as a supremum in the Smyth powerdomain ordering [22]. We will show that in a certain type of domains multivalued dependencies are in one-to-one correspondence with the decompositions of relations.

The model proposed in [5] does not admit constructions containing sets. However, they are necessary in order to describe some models which are being widely studied now, namely *nested relations* [6, 13, 16, 17, 18, 20] and *complex objects* [1, 13] which play an essential role in the theory of object-oriented databases [2]. By *complex objects* we mean objects constructed from the basis ones by using the operations of forming records (including discriminated union) and sets, i.e. record, variant and set constructors. This concept will cover all the structures that do not contain sets, and also nested relations and complex objects as they were defined in [1]. However, constructions containing sets can not be modeled by domains. In fact, we can not guarantee directedness. In this paper we will introduce a new concept generalizing domains (i.e. locally behaving as domains) which allows us to model complex objects. The schemes will be defined recursively.

When the concept of scheme is defined, we can easily define projection onto this scheme and selection. This is the crucial step in extending the ideas of relational algebra to generalized relations and complex objects.

This paper can be viewed as an extension of ideas of [5], that is, as the further development of domain-theoretic model of databases. The main contribution of [5] is the idea that relations can be generalized as finite antichains in domains. For such generalized relations a concept of scheme was introduced which allowed the development of some dependency theory (in fact, the analogy of functional dependency was introduced). A generalization of the join operation was also given.

In this paper we first examine the concept of scheme and give a new definition of scheme which expresses the fact that projections of complete descriptions onto a scheme are maximal among all possible projections. This definition is more general than that of [5] and, although the two definitions coincide if we speak of the domains of flat records (that is, of usual relations), there are some important cases when we need this new, more general definition. Moreover, the new definition satisfies almost all properties that were proved in [5]. When the concept of scheme is defined, we can introduce the operations of relational algebra for the generalized relations and prove some results about these operations. Then we define complements of schemes and use them to introduce multivalued dependencies for our domain model. The approach of [5] did not provide tools to work with complex objects or nested relations since they may contain sets whose sizes are not bounded *a priori*. We will extend the ideas of [5] to handle such constructions.

The paper is organized in five sections. The next section contains necessary definitions from domain theory [8, 21] and shows how relational databases can be represented and generalized in domains. The third section deals with schemes in domains. Two ways to introduce this concept will be discussed and the properties of schemes will be studied. In particular, we will characterize domains in which the two definitions coincide. Section 4 deals with structures that do not contain sets. We characterize schemes in these structures and introduce projection, selection and join. Having done this, we develop some dependency theory in section 5. Section 6 deals with the extension of our approach to complex objects. We will generalize the concept of domain in order to model these structures and then recursively defined schemes.

2 Powerdomain model of relational databases : Generalized relations

In denotational semantics of programming languages expressions denote values, and the domains of values are partially ordered. A database is a collection of objects having descriptions and meanings. The meaning is the set of all possible objects described by a description. The meaning having been defined as sets, we can order descriptions by saying that a description d_1 is better than a description d_2 if it describes less objects, i.e. if it is a more precise description.

Let $[\![d]\!]$ stand for the meaning of d. Suppose that d_1 and d_2 are the records in a relational database and

$$d_1 = \{\text{Dept} \Rightarrow '\text{CIS}', \text{Office} \Rightarrow '176'\},$$

$$d_2 = \{\text{Name} \Rightarrow '\text{Howard}', \text{Dept} \Rightarrow '\text{CIS}', \text{Office} \Rightarrow '176'\}.$$

Assume that there are no attributes except for name, department and office. Then the meaning of d_1 is the set of all possible records that refer to CIS people in office 176, in particular, d_2. Therefore, d_2 is better than d_1 because $[\![d_2]\!] \subseteq [\![d_1]\!]$.

The above ordering corresponds to the usual one in the theory of databases with incomplete information, in fact, to the ordering of tuples of Codd tables [10]. This approach is based on the assumption that we do not distinguish two different occurrences of null values in contrast to the approach of [4]. The same idea of ordering was used for complex objects in [1].

Suppose that the records in a relational database are described as functions from $\mathcal{L}$ to $\mathcal{V}_\perp$ where $\mathcal{L}$ is a set of attributes (in the above example $\mathcal{L} = \{\text{Name}, \text{Dept}, \text{Office}\}$) and $\mathcal{V}_\perp$ is a domain of values which is partially ordered. Then the records are also partially ordered by $d_1 \leq d_2$ iff $d_1(l) \leq d_2(l)$ for all $l \in \mathcal{L}$ where $d_1, d_2 : \mathcal{L} \to \mathcal{V}_\perp$.

Let $\mathcal{V}_\perp = \mathcal{V} \cup \{\perp\}$ where $\perp$ corresponds to incomplete information and $\forall v \in \mathcal{V} : \perp \leq v$ while all elements of $\mathcal{V}$ are incomparable. The set of functions from $\mathcal{L}$ to $\mathcal{V}_\perp$, denoted by $\mathcal{L} \to \mathcal{V}_\perp$, is ordered according to the above rule. For example, if d_1 and d_2 are as in the above example, $\mathcal{L} = \{\text{Name}, \text{Dept}, \text{Office}\}$ and $\mathcal{V}$ contains names of departments, people and numbers of offices, then $d_1, d_2 \in \mathcal{L} \to \mathcal{V}_\perp$ since
$d_1 = \{\text{Name} \Rightarrow \perp, \text{Dept} \Rightarrow '\text{CIS}', \text{Office} \Rightarrow '176'\}$.
Obviously $d_1 \leq d_2$.

Let $\mathcal{D} = \mathcal{L} \to \mathcal{V}_\perp$. Then the ordering of $\mathcal{D}$ satisfies the following properties:
1) *Every nonempty subset of $\mathcal{D}$ has a greatest lower bound;*
2) *Every directed subset of $\mathcal{D}$ has a least upper bound;*
3) *The set $K(\mathcal{D})$ of compact elements of $\mathcal{D}$ forms a countable basis of $\mathcal{D}$.*[1]
A poset (partially ordered set) satisfying 1)-3) is called a *Scott-domain* [8, 21]. We do not use any other kind of domain, and we will write simply *domain* instead of Scott-domain.

Least upper and greatest lower bounds will be denoted by $\vee$ and $\wedge$ respectively. $\uparrow x$ and $\downarrow x$ are the principal filter and ideal of $x \in \mathcal{D}$, i.e. the set of all elements of $\mathcal{D}$ which are greater (less) than x. Given a domain $\mathcal{D}$, every element of $\mathcal{D}$ is bounded above by some element of $\mathcal{D}^{max}$, the set of maximal elements of $\mathcal{D}$ [8, 21]. Elements of $\mathcal{D}^{max}$ are thought of as being complete descriptions. Therefore $[\![d]\!] = \uparrow d \cap \mathcal{D}^{max}$.

A domain is called *distributive* iff every $\downarrow x$ is a distributive lattice. We will call a domain *qualitative* iff every $\downarrow x$ is a Boolean lattice[2].

A number of ways have been described in [5] to construct domains representing certain kinds of data structures. Consider the domain $\mathcal{L} \to \mathcal{V}_\perp$. Its elements are records whose attributes are elements of $\mathcal{L}$ and values are taken from $\mathcal{V}_\perp$. It means

[1]A subset of a poset is called *directed* if all its finite subsets have least upper bounds. An element a is called *compact* if $a \leq \bigvee X$ implies $a \leq x$ for some $x \in X$ for any directed X. A subset $K \subset \mathcal{D}$ is called a *basis* if every element of $\mathcal{D}$ is a supremum of some elements from K [3].

[2]It is not hard to show that this definition is equivalent to the usual definition of a qualitative domain [7].

that there are a countable number of values and a special symbol $\perp$ corresponding to incomplete information. The ordering of $\mathcal{V}_\perp$ is given by letting $\perp$ be less than any other value. The relations are finite sets of records, i.e. finite subsets of $\mathcal{L} \to \mathcal{V}_\perp$. However, not every finite subset of $\mathcal{L} \to \mathcal{V}_\perp$ corresponds to a relation. If we have a subset containing both d_1 and d_2 from our example, d_1 is less informative than d_1 and should be removed. Less informative here means that $d_1 \leq d_2$. Therefore, relations correspond to finite subsets of domains that do not contain comparable elements, i.e. to *antichains*. This gives us the main idea of the generalization of relational databases proposed in [5]: *Generalized relations are antichains in domains.*

Example 1 Let $\mathcal{L}$ and $\mathcal{V}$ be as in the above examples. Let

$$d_3 = \{\text{Name} \Rightarrow \text{'Katherine'}, \text{Dept} \Rightarrow \text{'SL'}, \text{Office} \Rightarrow \text{'628'}\},$$

$$d_4 = \{\text{Name} \Rightarrow \text{'Katherine'}, \text{Dept} \Rightarrow \text{'SL'}, \text{Office} \Rightarrow \perp\},$$

(d_4 shows that the person has not been assigned an office yet). Then $\{d_2, d_3\}$ is a generalized relation but both $\{d_1, d_2\}$ and $\{d_3, d_4\}$ are not since $d_1 \leq d_2$ and $d_4 \leq d_3$. □

We will call finite antichains in domains *relations*. By *relations without incomplete information* we mean finite antichains of maximal elements, i.e. relations containing only complete descriptions.

We have shown so far how to order records of relations. The next problem is to order relations themselves, i.e. to order finite antichains of domains. In domain theory three ways to do this have been proposed:

$$A \sqsubseteq^\flat B \text{ iff } \forall a \in A\ \exists b \in B : a \leq b$$

$$A \sqsubseteq^\sharp B \text{ iff } \forall b \in B\ \exists a \in A : a \leq b$$

$$A \sqsubseteq^\natural B \text{ iff } A \sqsubseteq^\flat B \text{ and } A \sqsubseteq^\sharp B$$

called respectively Hoare, Smyth and Egli-Milner orderings[3]. Sets of finite antichains of a domain ordered by $\sqsubseteq^\flat$ or $\sqsubseteq^\sharp$ are distributive lattices (however, they are not complete).

The ordering $\sqsubseteq^\flat$ was used in the theory of relations with incomplete information to construct so-called representation systems, see [10]. When applied to an element of domain and a relation, this ordering expresses the notion of "x-belong" used for representation relations with null values by extended relations, see [25]. It was also used to order complex objects in [1].

A downward closed subset of a domain $\mathcal{D}$ which is closed under existing joins is called *a strong ideal*[4]. If $\mathcal{I} \subseteq \mathcal{D}$ is a strong ideal, then $p_\mathcal{I}$ defined by

$$p_\mathcal{I}(x) = \bigvee \{y : y \leq x \text{ and } y \in \mathcal{I}\}$$

is a *projection*, i.e. it satisfies the following properties: for all $x, y \in \mathcal{D}$: $p_\mathcal{I}(x) \leq x$, $p_\mathcal{I}(p_\mathcal{I}(x)) = p_\mathcal{I}(x)$ and $x \leq y$ implies $p_\mathcal{I}(x) \leq p_\mathcal{I}(y)$. Moreover, $p_\mathcal{I}$ is the unique projection on $\mathcal{D}$ with image $\mathcal{I}$.

Strong ideals can be equivalently described via projections onto them or their sets of maximal elements. In the other words, there are one-to-one correspondences between sets of strong ideals $\mathcal{I} \subseteq \mathcal{D}$, projections $p_\mathcal{I}$ and antichains of maximal elements of $\mathcal{I}$.

Example 2 Let $\mathcal{L}$, $\mathcal{V}$ and d_i's be as in the above examples. Let

$$\mathcal{I}_1 = \{\{\text{Name} \Rightarrow v, \text{Dept} \Rightarrow \perp, \text{Office} \Rightarrow \perp\} \mid v \in \mathcal{V}_\perp\}.$$

Then $\mathcal{I}_1$ is a strong ideal and for any

$$d = \{\text{Name} \Rightarrow v_1, \text{Dept} \Rightarrow v_2, \text{Office} \Rightarrow v_3\}$$

its projection onto $\mathcal{I}_1$ is

$$p_{\mathcal{I}_1}(d) = \{\text{Name} \Rightarrow v_1, \text{Dept} \Rightarrow \perp, \text{Office} \Rightarrow \perp\}.$$

The set of maximal elements of $\mathcal{I}_1$ is $\{\{\text{Name} \Rightarrow v, \text{Dept} \Rightarrow \perp, \text{Office} \Rightarrow \perp\} \mid v \in \mathcal{V}\}$.

Let $\mathcal{I}_2 = \downarrow d$ where $d \in \mathcal{L} \to \mathcal{V}_\perp$. Then $\mathcal{I}_2$ is a strong ideal with unique maximal element d and for any $d' \in \mathcal{L} \to \mathcal{V}_\perp : p_{\mathcal{I}_2}(d') = d \wedge d'$. □

[3]The orderings $\sqsubseteq^\flat$ and $\sqsubseteq^\sharp$ are known from lattice theory [3].

[4]The term *strong ideal* was used in [5]. A more precise name would be *downward closed subdomain*, suggested by Carl Gunter. However, we follow the terminology of [5] here.

We need more for the analogy of projection in relational algebra than being a projection onto strong ideal. In fact, this ideal must satisfy some additional properties. In domain $\mathcal{L} \to \mathcal{V}_\perp$ schemes correspond to subsets of $\mathcal{L}$, i.e. a projection onto scheme corresponding to $S \subseteq \mathcal{L}$ is given by $p_S(x) = x'$ where $x'(l) = x(l)$ if $l \in S$ and $x'(l) = \perp$ otherwise. These projections will be called *canonical*. It is a natural requirement for the definition of scheme and projection in an arbitrary domain that the projections be canonical for domain $\mathcal{L} \to \mathcal{V}_\perp$. One can easily see that for every $x \in \mathcal{L} \to \mathcal{V}_\perp$ the ideal $\downarrow x$ is strong while the projection $p_{\downarrow x}$ is not canonical.

The *slide condition* was introduced in [5] in order to give a definition of scheme. A strong ideal $\mathcal{I}$ is said to satisfy the slide condition if for any $x \in \mathcal{D}$ and $y \in \mathcal{I}$, $p_\mathcal{I}(x) \leq y$ implies that $x \vee y$ exists. This property obviously holds for canonical projections in $\mathcal{L} \to \mathcal{V}_\perp$.

An antichain $S \subseteq \mathcal{D}$ was called a *scheme* in [5] if $\downarrow S = \bigcup_{x \in S} \downarrow x$ is a strong ideal satisfying the slide condition. It can be easily concluded from the results of [5] that all schemes of $\mathcal{L} \to \mathcal{V}_\perp$ are determined by canonical projections. In [5] the main properties of schemes were studied and the schemes were used to develop some dependency theory.

In the next sections we discuss in detail the concept of scheme and introduce an alternative definition which will allow us to prove most of the results from [5] and further develop the ideas of that paper. This will allow us to introduce the main operations of the relational algebra for generalized relations, the latter being generalizations of relations admitting null values, records and discriminated unions. Then we show how to generalize our main concepts for structures containing sets, i.e. complex objects.

3 Schemes in domains

The main aim of this section is to discuss the definition of scheme in domains. The relations having been interpreted as antichains in Scott-domains, the concept of scheme is necessary in order to introduce an operation analogous to projection in the relational algebra.

In the domain $\mathcal{L} \to \mathcal{V}_\perp$ schemes correspond to the subsets of $\mathcal{L}$ and projections to the canonical projections. It is natural to define the concept of scheme such that, being applied to $\mathcal{L} \to \mathcal{V}_\perp$, it will give rise exactly to canonical projections. Also, schemes should be significant parts of a domain which reflect the structure of the whole domain. It means that if the elements of a domain are treated as database objects (for example, records of relations) then projection into an ideal generated by a scheme should correspond to loosing some piece of information and the same pieces of information are lost for all the elements of the domain. This means that projections generated by schemes are in a way homogeneous.

If we have two maximal elements of a domain (complete descriptions) and they are projected into a scheme (i.e. the same pieces of information are ignored) then the projections can not be comparable. This observation leads us to the following definition.

Definition Let $\mathcal{D}$ be a domain and S an antichain in D such that $\downarrow S$ is a strong ideal. Then S is called a *scheme* in $\mathcal{D}$ if projection $p_{\downarrow S}(x)$ of any element of $x \in \mathcal{D}^{max}$ is a maximal element in $\downarrow S$.

It is not hard to see that it is enough to require that projection of two maximal elements of $\mathcal{D}$ be incomparable instead of requiring that they be maximal in the corresponding ideal. We need some more concepts.

Definition Let $S \subseteq \mathcal{D}$ be a scheme. Then $\downarrow S$ is called a *scheme-ideal* and $p_{\downarrow S}$ is called a *scheme-projection*. We will write p_S instead of $p_{\downarrow S}$.

In the reasonings that led us to the above definition we took into account only *how* we loose information projecting into a scheme. In [5] another aspect of the problem was considered : what can be said about the lost information? Can we consider it independently and "add" to another object (element of domain)?

The idea of [5] was that, given a scheme, there is its complement (as there is a complement $\mathcal{L} - S$ for every $S \subseteq \mathcal{L}$ for the domain $\mathcal{L} \to \mathcal{V}_\perp$), and projecting into scheme S is simply loosing information corresponding to the complement of S [5]. Assuming that the pieces of information contained in projections into the scheme and its complement are independent, we can combine them. To be more precise, if we have an object and its projection into a scheme is less than an element of this scheme, we can add lost information to the latter element. This is the idea of P.Buneman's definition of scheme. Since we have already used the word "scheme", we will use term *semi-factor* proposed in [12].

Definition [5, 12] Let $\mathcal{D}$ be a domain and S an antichain such that $\downarrow S$ is a strong ideal. Then S is called a *semi-factor* if $\downarrow S$ satisfies the slide condition, that is, given $x \in \mathcal{D}$ and $y \in \downarrow S$ such that $p_S(x) \leq y$, then $x \vee y$ exists. $\downarrow S$ is called a semi-factor ideal, and p_S is called a semi-factor projection.

Every semi-factor is a scheme; the converse is not true in general. If it were true, it would mean (informally) that for all the schemes their complements exist, because we could consider the paragraph before the definition of semi-factor as an informal proof. In a certain class of domains this can be formally proved, and we will finish this section with such a result.

Example 3 Let d_2, d_3 be as in the examples 1 and 2. Let

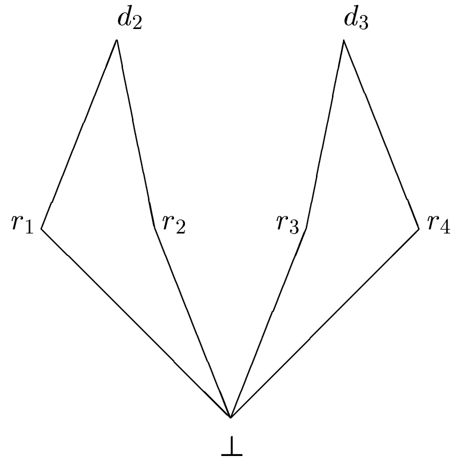
$$r_1 = \{\text{Name} \Rightarrow \text{'Howard'}, \text{Dept} \Rightarrow \perp, \text{Office} \Rightarrow \perp\},$$

$$r_2 = \{\text{Name} \Rightarrow \perp, \text{Dept} \Rightarrow \text{'CIS'}, \text{Office} \Rightarrow \text{'176'}\},$$

$$r_3 = \{\text{Name} \Rightarrow \text{'Katherine'}, \text{Dept} \Rightarrow \text{'SL'}, \text{Office} \Rightarrow \perp\},$$

$$r_4 = \{\text{Name} \Rightarrow \perp, \text{Dept} \Rightarrow \perp, \text{Office} \Rightarrow \text{'628'}\}.$$

Let $\mathcal{D} = \{d_2, d_3, r_1, r_2, r_3, r_4, \perp\}$ where $\perp$ is the tuple with all null values. The diagram of $\mathcal{D}$ is shown below:



This domain has no semi-factors but $\{\perp\}$ and $\mathcal{D}^{max}$ while it has eight proper schemes: $\{r_1, r_3\}$, $\{r_2, r_3\}$, $\{r_1, r_4\}$, $\{r_2, r_4\}$, $\{d_2, r_3\}$, $\{d_2, r_4\}$, $\{d_3, r_1\}$, $\{d_3, r_2\}$. □

In order to justify both definitions we must prove that they describe exactly canonical projections when applied to the domain $\mathcal{L} \to \mathcal{V}_\perp$.

Proposition 1 *S is a scheme (or a semi-factor) of $\mathcal{L} \to \mathcal{V}_\perp$ iff p_S is a canonical projection.* □

If $\mathcal{L}$ is finite, $\mathcal{L} \to \mathcal{V}_\perp$ is isomorphic to $\mathcal{V}_\perp^n$, where $n = |\mathcal{L}|$. Therefore, in direct products of flat domains all schemes are semi-factors. Theorem 3 below will generalize this fact.

We shall mostly use schemes rather than semi-factors because the definition of schemes is more general and does not make use of any additional assumptions, and, as we are going to show, schemes satisfy almost all properties that were proved in order to justify the definition of semi-factor in [5]. In the rest of the section we establish some necessary properties of schemes and state a

[5] In fact, it was not stated in [5] explicitly, but it seems to be the most natural interpretation of the slide condition.

result characterizing qualitative domains in which the concepts of scheme and semi-factor coincide.

Let $A, B \subseteq \mathcal{D}$ be two sets. We define $A \vee B$ as pointwise supremum, i.e. $A \vee B = \{a \vee b : a \in A, b \in B\}$.

Proposition 2 *Let $\mathcal{D}$ be a distributive domain. Then*
1) *If A, B are scheme-ideals, then so is $A \vee B$;*
2) *The set of scheme-ideals over $\mathcal{D}$ is a complete lattice.* □

The same results have been proved for semi-factors in [5]. Notice that scheme-ideals may not be closed under intersection in contrast to the case of semi-factor ideals. Proposition 2(2) says that schemes ordered by $\sqsubseteq^\flat$ form a lattice if $\mathcal{D}$ is distributive. A question arises : what can be said about other powerdomain orderings $\sqsubseteq^\sharp$ and $\sqsubseteq^\natural$? The following result shows that these orderings coincide for schemes in any domain. The same result for semi-factors was proved in [5].

Theorem 1 *Let $\mathcal{D}$ be an arbitrary domain and A, B two schemes. Then $A \sqsubseteq^\flat B$ iff $A \sqsubseteq^\sharp B$ iff $A \sqsubseteq^\natural B$.* □

Direct product ($\times$) and separated sum ($+$) are two important operations over domains. Direct product is defined as usual. Given two domains $\mathcal{D}_1$ and $\mathcal{D}_2$, $\mathcal{D} = \mathcal{D}_1 + \mathcal{D}_2$ is defined as follows : the set of its element is $(\mathcal{D}_1 \times \{1\}) \cup (\mathcal{D}_2 \times \{2\}) \cup \{\perp\}$, the ordering is inherited from the orderings of $\mathcal{D}_1$ and $\mathcal{D}_2$ and $\perp$ is the new bottom element. For example, a subdomain $\{d_2, d_3, r_2, r_3, \perp\}$ of the domain in example 3 (see the picture above) is isomorphic to $\mathcal{D}_1 + \mathcal{D}_2$ where $\mathcal{D}_1 = \{d_2, r_2\}$ and $\mathcal{D}_2 = \{d_3, r_3\}$. This construction corresponds to **case**, or discriminated union, while direct product corresponds to forming records. It is, therefore, important to describe schemes in products and sums.

Theorem 2 *Let $\mathcal{D} = \mathcal{D}_1 \times \mathcal{D}_2$ (or $\mathcal{D} = \mathcal{D}_1 + \mathcal{D}_2$). Then S is a scheme in $\mathcal{D}$ iff $S = S_1 \times S_2$ (or $S = S_1 + S_2$) for some schemes S_1 and S_2 in $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively.* □

In another paper I shall go further into mathematics of schemes; for our current purposes we do not need any more. We finish this section by the result describing qualitative domains in which the concepts of scheme and semi-factor coincide.

Theorem 3 (see also [12]). *Let $\mathcal{D}$ be a qualitative domain. Every scheme of $\mathcal{D}$ is a semi-factor iff*

$$\mathcal{D} \simeq \prod_{i \in I} \mathcal{D}_i$$

where each $\mathcal{D}_i$ has no proper scheme; the schemes of $\mathcal{D}$ are in 1-1 correspondence with subsets of I. □

4 Relational algebra for generalized relations

In this section we find the analogies of the main operations of relational algebra for generalized relations. Schemes introduced in the previous section will be used to define projections. The projections having been defined, we can introduce selection. The join operation will be borrowed from [5].

In order to construct generalized relations we can use the idea of [1]: starting with basic objects we use constructors such as *record* and *variant* (in this section we do not use *set*). Basic objects are elements of given domains, i.e. domains corresponding to basic types such as integers, characters etc. Generalized records are elements of domains obtained from the basic ones by using operation $\times$ for record constructor and $+$ for variant constructor. Generalized relations are finite sets of generalized records.

Let $\mathcal{B}$ be a set of domains. We now can recursively define domains of generalized records (equivalently, their types).

Definition (Database Domains)
1) Any $\mathcal{D} \in \mathcal{B}$ is a database domain;

2) (*record constructor*) If $\mathcal{D}_1, \ldots, \mathcal{D}_n$ are database domains, then $\mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ is a database domain; 3) (*variant constructor*) If $\mathcal{D}_1, \ldots, \mathcal{D}_n$ are database domains, the $\mathcal{D}_1 + \ldots + \mathcal{D}_n$ is a database domain.

Example 4 Let $\mathcal{B}$ contain three domains: $\mathcal{D}$, whose elements are people's names, $\mathbf{N}_\perp = \{\perp, n_1, n_2, ...\}$ representing natural numbers, and $\mathbf{Bool} = \{\perp, 0, 1\}$ representing booleans (in the above domains $\perp \leq n_i$ for all i and $\perp \leq 0, 1$). Suppose a database contains records with variants that have name and age fields for each person. If a person is a faculty member, the record contains his/her salary, and if he/she is a a student, it contains a subrecord indicating whether a student is supported and the amount of support. Below are the examples of such records:

$$r_1 = \{\text{Name} \Rightarrow \text{'John'}, \text{Age} \Rightarrow \text{'35'}, \text{Status} \Rightarrow \langle \text{Faculty} \Rightarrow \{\text{Salary} \Rightarrow \text{'40,000'}\}\rangle\},$$

$$r_2 = \{\text{Name} \Rightarrow \text{'Mary'}, \text{Age} \Rightarrow \text{'22'}, \text{Status} \Rightarrow \langle \text{Student} \Rightarrow \{\text{Supported} \Rightarrow \text{'0'}, \text{Amount} \Rightarrow \text{'0'}\}\rangle\},$$

$$r_3 = \{\text{Name} \Rightarrow \text{'Peter'}, \text{Age} \Rightarrow \text{'24'}, \text{Status} \Rightarrow \langle \text{Student} \Rightarrow \{\text{Supported} \Rightarrow \text{'1'}, \text{Amount} \Rightarrow \text{'12,000'}\}\rangle\}.$$

These records are elements of a database domain $\mathcal{D} \times \mathbf{N}_\perp \times (\mathbf{N}_\perp + (\mathbf{Bool} \times \mathbf{N}_\perp))$. □

Definition A *generalized record* is an element of a database domain. A *generalized relation* is a finite antichain in a database domain[6]. As we stated before, we will often omit the word "generalized".

For example, r_1, r_2, r_3 defined above are generalized records and $R = \{r_1, r_2, r_3\}$ is a (generalized) relation.

It is not hard to describe a type system using the given definition of database domains as it was done in [14, 15]. Suppose we have basic types τ_i^0 whose domains of values $\mathcal{D}_i^0$ are exactly domains from $\mathcal{B}$. Let $\mathcal{L}$ be a set of labels. Denote the domain of values of type τ by $[\![\tau]\!]$. Then if $\tau_1, \ldots, \tau_n$ are types, then so are $\{l_1 \Rightarrow \tau_1, \ldots, l_n \Rightarrow \tau_n\}$ and $\langle l_1 \Rightarrow \tau_1, \ldots, l_n \Rightarrow \tau_n \rangle$, where $l_1, \ldots, l_n \in \mathcal{L}$, and

$$[\![\{l_1 \Rightarrow \tau_1, \ldots, l_n \Rightarrow \tau_n\}]\!] = [\![\tau_1]\!] \times \ldots \times [\![\tau_n]\!],$$

$$[\![\langle l_1 \Rightarrow \tau_1, \ldots l_n \Rightarrow \tau_n \rangle]\!] = [\![\tau_1]\!] + \ldots + [\![\tau_n]\!].$$

Since domains are closed under direct product and separated sum, all database domains are domains. Therefore, we can speak of schemes in the database domains. There exists another way to define schemes using our recursive definition of database domains. Schemes in domains from $\mathcal{B}$ are just schemes as they were defined in the previous section; schemes in $\mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ are $S_1 \times \ldots S_n$ and schemes in $\mathcal{D}_1 + \ldots + \mathcal{D}_n$ are $S_1 + \ldots + S_n$ where S_i is a scheme in $\mathcal{D}_i$. According to theorem 2 these two definitions are equivalent.

Now we can define the operations of relational algebra for generalized relations. We will need one more definition: by $\min X$ and $\max X$ we will mean the sets of minimal and maximal elements of an ordered set X, respectively.

1. Union. Let $\mathcal{D}$ be a domain and R_1, R_2 two relations. Then their union $R_1 \tilde{\cup} R_2$ is $\max(R_1 \cup R_2)$.

We need the max operation because $R_1 \cup R_2$ may fail to be an antichain, but $R_1 \tilde{\cup} R_2$ always is. $R_1 \tilde{\cup} R_2$ can be interpreted as the set of the most informative elements from R_1 and R_2.

2. Difference. Let $\mathcal{D}$ be a domain and R_1, R_2 two relations. Then $R_1 - R_2$ is the usual set difference. Since $R_1 - R_2 \subseteq R_1$, it is a relation.

Intersection can be expressed as $R_1 \cap R_2 = R_1 - (R_1 - R_2)$.

3. Cartesian (direct) product. Let $\mathcal{D}_1, \mathcal{D}_2$ be two domains and R_1, R_2 relations in $\mathcal{D}_1, \mathcal{D}_2$ respectively. Then $R_1 \times R_2$ is a relation in $\mathcal{D}_1 \times \mathcal{D}_2$ defined as $\{\langle r_1, r_2 \rangle \mid r_1 \in R_1, r_2 \in R_2\}$.

[6] Therefore a generalized relation consists of objects of the same type as it is in the case of relational databases if a database is just a relation. It is not, however, a restriction for if we have objects of different types we can always use either variant or record constructor and consider these objects as having the same type.

4. Projection. Given a (database) domain $\mathcal{D}$, we define *projection* as projection into a scheme-ideal $\downarrow S$ in $\mathcal{D}$. If $\mathcal{D}$ is $\mathcal{L} \to \mathcal{V}_\perp$, then projections thus defined coincide with projections in relational algebra.

If $R \subseteq \mathcal{D}$ is a relation and S is a scheme, $p_S(R)$ may fail to be an antichain. Therefore, we need two operations of projection:

$$p_S^{min}(R) = \min p_S(R), \;\; p_S^{max}(R) = \max p_S(R).$$

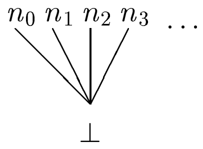
If R is a one-element relation, these two projections coincide and we will write simply $p_S(R)$. The above defined operations also coincide for relations without incomplete information, i.e. subsets of $\mathcal{D}^{max}$.

5. Selection. We can also define *selection* using the concept of scheme. First we have to define conditions. As usually, if c_1, c_2 are conditions, then so are $c_1 \vee c_2, c_1 \& c_2$ and $\neg c_1$. Schemes are necessary to define conditions we start with. Let $S, S' \subseteq \mathcal{D}$ be schemes, $a \in \downarrow S$, $x \in \mathcal{D}$. Then the elementary conditions are $p_S(x)\theta a$, $p_S(x)\theta p_{S'}(x)$, where $\theta \in \{<, \leq, =, \neq, \geq, >\}$.

Let $R \subseteq \mathcal{D}$ be a relation. i.e. an antichain in $\mathcal{D}$. If $c : \mathcal{D} \to \{\mathbf{T}, \mathbf{F}\}$ is a condition, then the *selection* is defined as $\sigma_c(R) = \{x \in R : c(x) = \mathbf{T}\}$.

If we do not know what the basic domains from $\mathcal{B}$ are and how $\mathcal{D}$ was constructed from them, the above defined selection is all we can get. However, if we know a concrete procedure of construction of $\mathcal{D}$ (i.e. a term in signature $\langle \times, + \rangle$ with variables from $\mathcal{B}$) then we can define more complex conditions. For example, if the database domain is $\mathcal{D} \times \mathcal{D} \times \mathcal{D}$ we may want to select those element whose first and third projections coincide.

We can give the selection more power if we introduce binary relations on domains from $\mathcal{B}$. For example, if P is a binary relation on $\mathcal{D}_1 \in \mathcal{B}$ and $\downarrow S = \mathcal{D}_1$, then we can introduce conditions like $(p_S(x), a) \in P$. This is necessary because, for example, domain of natural numbers is represented in domain theory as $\mathbf{N}_\perp = \{\perp, n_0, n_1, n_2, \ldots\}$ where n_i corresponds to the natural number i, and the ordering of $\mathbf{N}_\perp$ is given by letting $\perp$ be less than all n_i's:



$\mathbf{N}_\perp$

We can not conclude that $1 < 2$ from this information, therefore, we need a binary relation P on $\mathbf{N}_\perp$ describing the ordering of natural numbers.

To define such powerful selection we first need the definition of *similar* schemes and a 1-1 correspondence between their scheme-ideals. In the above example of $\mathcal{D} \times \mathcal{D} \times \mathcal{D}$ schemes $\mathcal{D} \times \{\perp\} \times \{\perp\}$ and $\{\perp\} \times \{\perp\} \times \mathcal{D}$ should be similar and 1-1 correspondence between their scheme-ideals associates the first and the third projections of any record. This gives us possibility to compare projections on different schemes. As it was said earlier, we may want, for example, to select records with coinciding first and third projections.

Given a database domain $\mathcal{D}$, it can be represented as $t(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ where t is a term of signature $\langle \times, + \rangle$ and $\mathcal{D}_1, \ldots, \mathcal{D}_n \in \mathcal{B}$ (e.g. the database domain in example 4 is $\mathcal{D} \times \mathbf{N}_\perp \times (\mathbf{N}_\perp + (\mathbf{Bool} \times \mathbf{N}_\perp)))$. We now define *similarity* of two schemes S, S' and mapping $\varphi_{S \to S'} : \downarrow S \to \downarrow S'$.

If S is a scheme in $\mathcal{D} \in \mathcal{B}$, then S is similar to itself and $\varphi_{S \to S}$ is the identical mapping on $\downarrow S$.

Let $\mathcal{D} = t(\mathcal{D}_1, \ldots, \mathcal{D}_n)$, where $\mathcal{D}_i \in \mathcal{B}$, $i = 1, \ldots, n$. Suppose S, S' are two schemes in $\mathcal{D}$. Let the last operation of t be $\times$, i.e. $t(\cdot, \ldots, \cdot) = t_1(\cdot, \ldots, \cdot) \times \ldots \times t_k(\cdot, \ldots, \cdot)$ and the last operation of each t_i is not $\times$. Then $S = S_1 \times \ldots \times S_k$ and $S' = S'_1 \times \ldots \times S'_k$ where S_i, S'_i are schemes in $t_i(\mathcal{D}_1, \ldots, \mathcal{D}_n)$, see theorem 2. S is similar to S' iff there are such i and j that $t_i = t_j$, S_i is similar to S'_j in $t_i(\mathcal{D}_1, \ldots, \mathcal{D}_n) = t_j(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ and $S_l = \{\perp_{t_l(\mathcal{D}_1, \ldots, \mathcal{D}_n)}\}$, $S'_p = \{\perp_{t_p(\mathcal{D}_1, \ldots, \mathcal{D}_n)}\}$, $l \neq i, p \neq j$. $\varphi_{S \to S'}$ maps a record $x \in \downarrow S$ with only nonbottom ith component $x_i \in \downarrow S_i$ to the record whose only nonbottom jth component is

$\varphi_{S_i \to S'_j}(x_i)$.

If the last operation of the term is $+$, then $S = S_1 + ... + S_k$ and $S' = S'_1 + ... + S'_k$ where S_i, S'_i are schemes in $t_i(\mathcal{D}_1, \ldots, \mathcal{D}_n)$. Then S is similar to S' iff each S_i is similar to S'_i in $t_i(\mathcal{D}_1, \ldots, \mathcal{D}_n)$, and for any $x \in \downarrow S : \varphi_{S \to S'}(x) = \varphi_{S_i \to S'_i}(x)$ if $x \in S_i$.

Example 5 Let $S = \{\perp\} \times \{\perp\} \times \mathcal{D}$ and $S' = \mathcal{D} \times \{\perp\} \times \{\perp\}$ be two schemes in $\mathcal{D} \times \mathcal{D} \times \mathcal{D}$. Then S and S' are similar and $\varphi_{S \to S'}(\{\perp, \perp, x\}) = \{x, \perp, \perp\}$.

Schemes $\mathcal{D} + (\{\perp\} \times \mathcal{D})$ and $\mathcal{D} + (\mathcal{D} \times \{\perp\})$ are similar in $\mathcal{D} + (\mathcal{D} \times \mathcal{D})$. □

Now we can extend the list of possible elementary conditions by adding the conditions of form $\varphi_{S \to S'}(p_S(x))\theta p_{S'}(x)$ where S, S' are two similar schemes in a database domain $\mathcal{D}$.

As we said before, one may also want to define some binary relations on basic domains. Let $P_i^k, k \in I_i$ be a family of binary relations on $\mathcal{D}_i \in \mathcal{B}$, where I_i is (possibly empty) set of indices. We say that a scheme S of a database domain $\mathcal{D} = t(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ is also a scheme in a basic domain $\mathcal{D}_i$ if $S = t(\{\perp\}, \ldots, S_i, \ldots, \{\perp\})$ where $S_i \subseteq \mathcal{D}_i$ is a scheme. In this case we can identify elements of $\downarrow S$ and $\downarrow S_i$.

The third type of elementary conditions includes the conditions $(p_S(x), a) \in P_i^k$ and $(p_S(x), p_{S'}(x)) \in P_i^k$ where S, S' are schemes in $\mathcal{D}_i$ identified with S_i, $a \in S_i$ and $k \in I_i$.

With such extensions being added, selection covers usual selection in relational algebra.

Example 6. Consider a relation with variants describing companies. Each record contain the following information: name, total donations for non-profit companies, gross revenue and costs for profit companies. Below are the examples of records:

$$r_1 = \{\text{Name} \Rightarrow '\text{X}', \text{Status} \Rightarrow \langle \text{Non} - \text{profit} \Rightarrow \{\text{Donations} \Rightarrow '1{,}000{,}000'\}\rangle\},$$

$$r_2 = \{\text{Name} \Rightarrow '\text{Y}', \text{Status} \Rightarrow \langle \text{Profit} \Rightarrow \{\text{Revenue} \Rightarrow '2{,}000{,}000', \text{Costs} \Rightarrow '1{,}000{,}000'\}\rangle\}.$$

Let $\mathcal{D}$ be a domain of names. Then the above records are elements of a database domain $\mathcal{D} \times (\mathbf{N}_\perp + (\mathbf{N}_\perp \times \mathbf{N}_\perp))$. Consider the following schemes:

$$S_1 = \{\perp_\mathcal{D}\} \times (\mathbf{N}_\perp + (\{\perp_{\mathbf{N}_\perp}\} \times \{\perp_{\mathbf{N}_\perp}\})),$$

$$S_2 = \{\perp_\mathcal{D}\} \times (\{\perp_{\mathbf{N}_\perp}\} + (\mathbf{N}_\perp \times \{\perp_{\mathbf{N}_\perp}\})),$$

$$S_3 = \{\perp_\mathcal{D}\} \times (\{\perp_{\mathbf{N}_\perp}\} + (\{\perp_{\mathbf{N}_\perp}\} \times \mathbf{N}_\perp)).$$

Then S_1, S_2, S_3 are also schemes in $\mathbf{N}_\perp$ and S_2 is similar to S_3.

Let P be a binary relation on $\mathbf{N}_\perp$ such that $(n_i, n_j) \in P$ iff $i \leq j$, $(\perp, x) \in P$ for all $x \in \mathbf{N}_\perp$. Consider the following conditions: $c_1 \equiv (p_{S_1}(x) \neq \perp_{\mathbf{N}_\perp})$ [7], $c_2 \equiv ((p_{S_3}(x), p_{S_2}(x)) \in P)$. Let R be a relation in the above database domain. Then $\sigma_{c_1}(R)$ selects non-profit companies from R while $\sigma_{c_2}(R)$ selects companies working well, that is, whose gross revenue exceeds costs. □

6. Join. Join was introduced in [5] as the supremum in Smyth powerdomain ordering, i.e., given two relations (antichains) $R_1, R_2 \subseteq \mathcal{D}$, their join is $R_1 \sqcup^\sharp R_2$. It was proved that for domain $\mathcal{L} \to \mathcal{V}_\perp$ the above defined operation coincides with the usual join in relational algebra, see [5]. We will write more convenient and customary symbol $\bowtie$ instead of $\sqcup^\sharp$.

There is another way to think of the join operation. Given two generalized relations $R_1, R_2 \subseteq \mathcal{D}$, their join $R_1 \bowtie R_2$ is the set of minimal (in $\mathcal{D}$) elements which are greater than some element of R_1 *and* some element of R_2 : $R_1 \bowtie R_2 = \min\{x \in \mathcal{D} \mid \exists r_1 \in R_1, r_2 \in R_2 : r_1 \leq x, r_2 \leq x\}$.

Several conditions were given in [23] that the analogy of the natural join in object-oriented model should satisfy. Informally, they are: 1) if there are no common attributes of two relations, the result of join is isomorphic to their direct (Cartesian) product; 2) if two relations are defined over

[7] To be more precise, we should compare $p_{S_1}(x)$ with an element of $\downarrow S_1$, that is, with $\{\perp_\mathcal{D}, \perp_{\mathbf{N}_\perp} \times \{1\}\}$.

the same sets of attributes, the result of join is their intersection; 3) the join of two relations can be obtained as the union of pairwise joins of its elements(where these exist). Join is also known to be associative in relational algebra, see [24].

Let us formalize the above properties.

1) *Let $R_1 \subseteq \mathcal{D}_1, R_2 \subseteq \mathcal{D}_2$ be two relations, and $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$. Let $R_1' = R_1 \times \{\perp_2\}$ and $R_2' = R_2 \times \{\perp_1\}$ be two relations in $\mathcal{D}_1 \times \mathcal{D}_2$. Then $R_1' \bowtie R_2' = R_1 \times R_2$.*

2) *Let $R_1, R_2 \subseteq \mathcal{D}^{max}$ be two relations. Then $R_1 \bowtie R_2 = R_1 \cap R_2$.*

Formalizing property 3) we must keep in mind that the union of pairwise joins may contain comparable elements while relations are antichains. Therefore, after finding union of joins we have to eliminate some elements in order to obtain an antichain. According to [10], there is no "semantically correct" way to do it. Since joining relations with null values may often yield counter-intuitive results (cf. [10, 14]) we think that formalizing the third property we have to eliminate nonminimal elements, i.e. to leave the least informative elements among pairwise joins.

3) *Let $R, R' \subseteq \mathcal{D}$ be two relations, and $R = \{r_1, \ldots, r_n\}, R' = \{r_1', \ldots, r_m'\}$. Then $R \bowtie R' = \min(\bigcup(\{r_i\} \bowtie \{r_j'\} : i = 1, ..., n, j = 1, ..., m))$.*

4) *If $R_1, R_2, R_3 \subseteq \mathcal{D}$ are three relations, then $R_1 \bowtie (R_2 \bowtie R_3) = (R_1 \bowtie R_2) \bowtie R_3$.*

Proposition 3 *The above defined join operation $\sqcup^\sharp$ satisfies 1) - 4).* □

It is known that in relational algebra join can be expressed via projection, selection and Cartesian product. This is not true for generalized relations. However, if the underlying domain is the direct product of domains then such a representation for join exists. Let $\mathcal{D} = \mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ and R_1, R_2 be two relations in $\mathcal{D}$. For any $x \in \mathcal{D}$ by x_i we mean its ith component, i.e. projection to $\mathcal{D}_i$. Let $R \subseteq \mathcal{D}$ be a relation, and $I(R) = \{i \mid \exists r \in R : r_i \neq \perp_{\mathcal{D}_i}\}$. Let $S_i = \{\perp\} \times \ldots \times \mathcal{D}_{k(i)} \times \ldots \times \{\perp\}$ where $k(i) = i$ if $i \leq n$ and $n - i$ otherwise and $\mathcal{D}_{k(i)}$ is the ith factor among $2n$ factors. Then S_i is a scheme in $\mathcal{D} \times \mathcal{D}$. Let S be the direct product of such S_is that $i \in I(R_1)$ for $i \leq n$ and $i - n \notin I(R_1)$ for $i > n$. Let c be the conjunction of conditions $p_{S_i}(x) = p_{S_{n+i}}(x)$ for all $i \in I(R_1) \cap I(R_2)$. Then

$$R_1 \bowtie R_2 = p_S^{min}(\sigma_c(R_1 \times R_2)).$$

We finish this section by showing that the above defined operations form an algebra, that is, generalized relations are closed under union $\tilde{\cup}$, difference, Cartesian product, projections, selection and join.

Theorem 4 *Generalized relations are closed under the operations $\tilde{\cup}, -, \times, p^{min}, p^{max}, \sigma, \bowtie$.* □

5 Dependency theory for generalized relations

Having introduced the notion of scheme, we can define functional dependencies. If S_1, S_2 are schemes in a domain $\mathcal{D}$, then a functional dependency is an expression of the form $S_1 \to S_2$. Usually in the theory of databases with incomplete information dependencies are defined only on the schemes projections on which do not contain tuples with null values. This condition can be equivalently expressed as: for any record in a relation there is a record in a scheme which is less informative than the relation record. In other words, if R is a relation and S is a scheme, then $S \sqsubseteq^\sharp R$.

Now we can define satisfiability for functional dependencies. Let $R \subseteq \mathcal{D}$ be a relation. We say that R satisfies functional dependency $S_1 \to S_2$ if $S_1, S_2 \sqsubseteq^\sharp R$ and $p_{S_2}(x) = p_{S_2}(y)$ whenever $p_{S_1}(x) = p_{S_1}(y)$ for every $x, y \in R$.

Functional dependencies in distributive domains have been investigated in [5] for the particular case of semi-factors, and the following analogies of the Armstrong axioms are due to [5], where F is a set of functional dependencies, Schemes($\mathcal{D}$) the complete lattice of schemes over distributive domain $\mathcal{D}$ (cf. proposition 2).

(a) If $S_1, S_2 \in$ Schemes($\mathcal{D}$) and $S_1 \leq S_2$ then $S_2 \to S_1 \in F$;

(b) If for any $i \in I : S \to S_i \in F$ where $S, S_i \in$ Schemes($\mathcal{D}$) then $S \to \bigvee_{i \in I} S_i \in F$;

(c) If $S_1 \to S_2 \in F$ and $S_2 \to S_3 \in F$, where $S_1, S_2, S_3 \in$ Schemes($\mathcal{D}$) then $S_1 \to S_3 \in F$.

The result of [5] proved for semi-factors is also true for schemes:

Proposition 4 *The Armstrong Axioms (a)–(c) are consistent and complete for relations in distributive domains.* □

Now our purpose is to introduce multivalued dependencies for generalized relations. A multivalued dependency $X \twoheadrightarrow Y$, where X, Y are sets, appeals to projection onto the set $X \cup \overline{Y}$. While $\cup$ corresponds to $\vee$ for domain model, there is no analogy for complement. More precisely, the poset of schemes is a lattice if the domain is distributive, but schemes may fail to have complements in contrast to the case of $\mathcal{L} \to \mathcal{V}_\perp$. Thus, two problems will be discussed in the rest of this section. The first one is how we can define complements. The complements having been defined, we introduce multivalued dependencies and prove a decomposition theorem.

Consider the domain $\mathcal{L} \to \mathcal{V}_\perp$. Its schemes correspond to subsets of $\mathcal{L}$, with scheme-projections being canonical projections. The complement of a scheme corresponds to projecting onto the complementary subset of $\mathcal{L}$.

Suppose that we have defined the concept of complement, p is a scheme-projection and $\overline{p}$ the projection corrsponding to the scheme's complement. What should the properties of $\overline{p}$ be? First, if we have any element $x \in \mathcal{D}$, then $p(x) \wedge \overline{p}(x) = \perp$. Suppose that $x \in \mathcal{D}^{max}$. Then $\overline{p}(x)$ "forgets" about information contained in $p(x)$. The fact that $\overline{p}$ is the complement of p means that all information contained in x can be reconstructed from $p(x)$ and $\overline{p}(x)$, i.e. $x = p(x) \vee \overline{p}(x)$. That means that in order to introduce complements, we have to require that all principal ideals $\downarrow x$ in $\mathcal{D}$ be complemented lattices. Moreover, they must be uniquely complemented since we want to speak about *the* complement. The next result easily follows from [19].

Proposition 5 *Any principal ideal of a domain $\mathcal{D}$ is a uniquely complemented lattice iff $\mathcal{D}$ is a qualitative domain.* □

Let $\mathcal{D}$ be a qualitative domain and $S \subseteq \mathcal{D}$ be a scheme. Consider the set $\overline{\mathcal{I}}_S = \{\overline{p}_S(x) : x \in \mathcal{D}^{max}\}$, where $\overline{p}_S(x)$ is the complement of $p_S(x)$ in $\downarrow x$. We would like $\overline{\mathcal{I}}_S$ to be the complement of S. However, it can be easily shown that $\overline{\mathcal{I}}_S$ may fail to be a scheme although $\downarrow \overline{\mathcal{I}}_S$ is always a strong ideal.

There is another elegant way to define complement proposed by A. Jung [11]. Let $S \subseteq \mathcal{D}$ be a scheme in any domain $\mathcal{D}$. We define $\mathcal{I}_{\overline{S}}$ as the set of maximal elements of $\{x \in \mathcal{D} : p_S(x) = \perp\}$. It also can be shown that $\mathcal{I}_{\overline{S}}$ is not generally a scheme. In order to be able to operate with complements, we have to make two observations.

Proposition 6 *Let $\mathcal{D}$ be a qualitative domain and S any scheme. Then $\downarrow \overline{\mathcal{I}}_S = \downarrow \mathcal{I}_{\overline{S}}$, i.e. $\mathcal{I}_{\overline{S}}$ is the set of maximal elements of $\overline{\mathcal{I}}_S$.* □

Given a scheme S in a qualitative domain, we can correctly define its complement as $\mathcal{I}_{\overline{S}}$. As we mentioned above, the complement of a scheme may not be a scheme. However, complements of semi-factors are schemes, as the following result shows.

Proposition 7 *The complement of a semi-factor is a scheme in any qualitative domain.* □

If $\mathcal{I}_{\overline{S}}$ is a scheme, we say that S has a complement (which is $\mathcal{I}_{\overline{S}}$) and denote it by $\overline{S}$.

Definition Let $\mathcal{D}$ be a qualitative domain and S a scheme having the complement $\overline{S}$. Let S' be a scheme. We say that a relation $R \subseteq \mathcal{D}$ satisfies *multivalued dependency* $S' \twoheadrightarrow S$ if for every $x, y \in R$ with $p_{S'}(x) = p_{S'}(y)$ there exists $z \in R$ such that $p_{S'}(z) \vee p_S(z) = p_{S'}(x) \vee p_S(x)$ and $p_{S'}(z) \vee p_{\overline{S}}(z) = p_{S'}(y) \vee p_{\overline{S}}(y)$.

If $\mathcal{D}$ is $\mathcal{L} \to \mathcal{V}_\perp$, we obtain the usual definition of multivalued dependency in a relational database. Notice that, like functional dependencies, multivalued dependencies should be considered only on schemes the projections into which do not contain null values. As it was shown above, it means that a scheme is less than a relation in Smyth powerdomain ordering $\sqsubseteq^\sharp$. Therefore in the above definition the following should hold: $S' \vee S \sqsubseteq^\sharp R$ and $S' \vee \overline{S} \sqsubseteq^\sharp R$. It can be easily concluded from the above inclusions that $R \subseteq \mathcal{D}^{max}$. Therefore we will consider only relations without incomplete information when speaking of multivalued dependencies.

The above introduced functional and multivalued dependencies satisfy two well-known properties:

Proposition 8 *Let $\mathcal{D}$ be a qualitative domain, and S a scheme having complement $\overline{S}$. Let S' be a scheme, and R a relation without incomplete information, i.e. a finite subset of $\mathcal{D}^{max}$. Then*
1) If R satisfies $S' \to S$ then R satisfies $S' \twoheadrightarrow S$;
2) If R satisfies $S' \twoheadrightarrow S$, then R satisfies $S' \twoheadrightarrow \overline{S}$. □

We have defined so far multivalued dependencies and the join operation. We also have shown that the complement of a semi-factor in a qualitative domain is a scheme. Now we are ready to formulate a decomposition theorem.

Theorem 5 *Let $\mathcal{D}$ be a qualitative domain, and R a relation without incomplete information (that is, a finite subset of $\mathcal{D}^{max}$). Let S' be a scheme and S a semi-factor of $\mathcal{D}$. Then R satisfies multivalued dependency $S' \twoheadrightarrow S$ iff $R = [p_{S'} \vee p_S(R)] \bowtie [p_{S'} \vee p_{\overline{S}}(R)]$, where join $\bowtie$ is $\sqcup^\sharp$.* □

We did not indicate which operation of projection – p^{min} or p^{max} – was used because they coincide for generalized relations without incomplete information.

6 Extending relational algebra to complex objects

The standard approach to constructing complex objects is to apply record, variant and set constructors to basic types. The crucial point is that we admit set constructor, i.e. given any type τ, there is a type $\{\!\{\tau\}\!\}$ whose instances are finite sets of objects of type τ. Thus, we can not use domains anymore, because we may have an increasing infinite chain of finite sets, which itself is not directed. In order to develop a "domain-like" theory for complex objects, we need to generalize the concept of domain. This new concept should be more general than that of domain. Moreover, the new objects we are going to define must be closed with respect to application of record, variant and set constructors.

Due to the limitations set up for the papers in this volume we are unable to present all details of the extension of the algebra from section 4 to constructions containing sets. Instead, we will give here the analogies of the main definitions which were in the focus of the first three sections, that is, the definitions of local domains, which are the generalization of domains that we are going to use, database domains, schemes and projections. Notice that the crucial steps in the defining algebra for generalized relations were to define generalized relations as finite antichains in domains, schemes and projections. In this section these main steps will be gone through in the case of complex objects.

If we allow a type $\{\!\{\tau\}\!\}$, then we allow an infinite sequence of sets $\{x_1\} \sqsubseteq^\flat \{x_1, x_2\} \sqsubseteq^\flat$

$\{x_1, x_2, x_3\} \sqsubseteq^\flat \ldots$, where all $x_i \in [\![\tau]\!]$. This sequence is a directed set but it does not have the least upper bound among instances of type $\{\{\tau\}\}$. For example, τ may be a record type and the instances of $\{\{\tau\}\}$ are relations, i.e. finite sets of records. If $\{\{\tau\}\}$ is used as a constructor for another record type, that is, if we deal with nested relations, then we may have infinite increasing sequence of higher-order records. However, if we are given any higher-order relation, i.e. a finite set of higher-order records, and a directed set below this relation, then this directed set has the least upper bound. Therefore, the higher-order records range over the poset which locally behaves as a domain. The following definition captures this property.

Definition A poset $\mathcal{D}$ is called a *local domain* if it satisfies the following properties:
1) $\mathcal{D}$ does not have infinite decreasing chains;
2) The set $K(\mathcal{D})$ of compact elements of $\mathcal{D}$ forms a countable basis of $\mathcal{D}$;
3) For any finite antichain $A \subseteq \mathcal{D}$, $\downarrow A$ is a domain.

We changed the requirement that $\mathcal{D}$ be directed in the definition of domain to the requirement that $\mathcal{D}$ be locally directed, i.e. every finite antichain should generate a domain. We also require that $\mathcal{D}$ be a poset without infinite decreasing chains. This condition guarantees that $\mathcal{D}$ is a complete semilattice, that is, every nonempty subset of $\mathcal{D}$ has the greatest lower bound. This condition is not a severe limitation since usually domains of basic types satisfy it and, as we are going to show in this section, it is preserved when record, variant and set constructors are applied (see theorem 6 below).

Definition (Database Domain) Let $\mathcal{B}$ be a set of basic local domains. The database domains are defined as follows:
1) Any local domain from $\mathcal{B}$ is a database domain;
2) If $\mathcal{D}_1, \ldots, \mathcal{D}_n$ are database domains, then so is $\mathcal{D}_1 \times \ldots \times \mathcal{D}_n$;
3) If $\mathcal{D}_1, \ldots, \mathcal{D}_n$ are database domains, then so is $\mathcal{D}_1 + \ldots + \mathcal{D}_n$;
4) If $\mathcal{D}$ is a database domain, then so is $\mathcal{P}(\mathcal{D})$ which is the set of finite subset of $\mathcal{D}$ ordered by $\sqsubseteq^\flat$, i.e. $\langle P_f(\mathcal{D}), \sqsubseteq^\flat \rangle$.

A *complex object* (to be more precise, *generalized* complex object) is defined as a finite antichain in a local domain.

This definition is more general than that used in [17, 18] where algebra, calculus and null values for nested relations were discussed. The first generalization is that we start with arbitrary local domains or domains. The variant constructor is also allowed, and each attribute can be relation-valued, that is, each domain used for forming records can be obtained as $\mathcal{P}(\mathcal{D})$ for some $\mathcal{D}$, while in [17, 18] it is assumed that the nested relations must be in partitioned normal form[8].

We need the following result about local domains.

Theorem 6 *Any domain without infinite decreasing chains is a local domain. If $\mathcal{D}_1, \mathcal{D}_2$ are local domains, then so are $\mathcal{D}_1 \times \mathcal{D}_2$, $\mathcal{D}_1 + \mathcal{D}_2$ and $\mathcal{P}(\mathcal{D}_1)$.* □

Corollary 1 *Any database domain is a local domain.* □

We are ready to define scheme-ideals and scheme-projections if $\mathcal{B}$ consists only of domains (which are local domains). This is a natural assumption, because elements of $\mathcal{B}$ are domains we start with, i.e. domains of basic types like integers, characters etc.

Definition (Projections and Ideals in Database Domains)

1) Scheme-ideals and scheme-projections in elements of $\mathcal{B}$ are just those in domains $\mathcal{D} \in \mathcal{B}$.

2) If $\mathcal{I}_1 \subseteq \mathcal{D}_1$ and $\mathcal{I}_2 \subseteq \mathcal{D}_2$ are scheme-ideals in local domains $\mathcal{D}_1$ and $\mathcal{D}_2$ and p_1, p_2 are corresponding scheme-projections, then $\mathcal{I}_1 \times \mathcal{I}_2$ is a scheme-ideal in $\mathcal{D}_1 \times \mathcal{D}_2$, with $p(\langle x_1, x_2 \rangle) = \langle p_1(x_1), p_2(x_2) \rangle$ being the corresponding scheme-projection.

[8]It means that zero order attributes form a key, and each nested subrelation of a less order must also be in partitioned normal form. In our model it can be the case that there are no zero order attributes.

3) If $\mathcal{I}_1 \subseteq \mathcal{D}_1$ and $\mathcal{I}_2 \subseteq \mathcal{D}_2$ are two scheme-ideals with the corresponding scheme-projections p_1 and p_2, then $\mathcal{I}_1 + \mathcal{I}_2$ is a scheme-ideal in $\mathcal{D}_1 + \mathcal{D}_2$, and for the corresponding scheme-projection p we have: $p(x) = p_i(x_i)$ if $x \in \mathcal{I}_i$, $i = 1, 2$.

4) Scheme-projections in $\mathcal{P}(\mathcal{D})$ are given by projections $P(\{x_1, \ldots, x_n\}) = \{p(x_1), \ldots, p(x_n)\}$ where p is a scheme-projection in $\mathcal{D}$. The corresponding scheme-ideal is $\{P(X) \,|\, X \in P_f(\mathcal{D})\}$.

Notice that we have not defined the schemes since a scheme-ideal in $\mathcal{P}(\mathcal{D})$ may not have the set of maximal elements. However, it was the concept of projection onto a scheme and not the one of scheme which was crucial for defining the operations of projection and selection in algebra.

If elements of a database domains are records of a nested relations, then the above defined scheme-projections are projections in the recursive algebra for nested relations of [6].

The definition of scheme-projections and ideals does take into account the way the database domain has been constructed, i.e. a term in signature $\langle \times, +, \mathcal{P} \rangle$ over variables from $\mathcal{B}$. Notice that the grammar-based approach to defining such terms and instances of the domains they correspond to was studied in [9].

7 Conclusion

In this paper we have been studying the new approach to generalization of relational datamodel that treats relations as subsets of domains, which are partially ordered sets of descriptions [5]. This approach allows us, for example, to model different ways of working with null values and records with variants.

In the paper we have described relational algebra operations for such generalized relations and outlined the ways of their extending to complex objects. We did it by defining *schemes* in domains and *projections* associated with the schemes. Join was defined as the supremum in a powerdomain ordering and it was shown to satisfy the analogies of the properties of natural join in relational algebra.

Functional and multivalued dependencies have been introduced and decomposition theorem establishing the relationship between join and multivalued dependencies has been proved for the generalized relations. In the case of complex objects it has been argued that domains can not serve as the basis of the model, and *local domains* have been defined to replace domains in the model and to make it possible to work with set constructor for complex objects. Recursive definitions of scheme-ideals and projections have been given.

Some of the open problems to which we would like to dedicate the further research are: constructing calculus associated with introduced algebra for generalized relations and complex objects; finding analogies of the basic concepts of relational database theory (for example, such as normalization) in our domain model; investigation of the different ways of treatment of null values from the domain theory point of view; extending the basic model in order to be able to operate with sets.

Acknowledgements This work was inspired by the idea of Peter Buneman to attract domain theory to generalize relational databases. I am very grateful to Peter Buneman and Achim Jung for very helpful discussions and suggestions. I also would like to thank Carl Gunter, Anthony Kosky and Val Tannen for their comments on this paper.

References

[1] F. Bancilhon, S. Khoshafin. A calculus for complex objects. In *PODS 1986.*

[2] C. Beeri. Formal models for object oriented databases. In : *Proc. of Int. Conf. on Deductive and Object-Oriented Databases*, Kyoto, December 1989.

[3] G. Birkhoff. *Lattice Theory.* 3rd ed., AMS, Providence, RI, 1967.

[4] J. Biskup. A formal approach to null values in database relations. In : *Advances in Database Theory* (H. Gallaire, J. Minker, J.M.. Nicolas, Eds.), Plenum Press, New York, 1981, pp.299–341.

[5] P. Buneman, A. Jung, A. Ohori. Using powerdomains to generalize relational databases. *Theoretical Computer Science*, 1991, to appear.

[6] L.S. Colby. A recursive algebra and query optimization for nested relations. In *SIGMOD 89.*

[7] J.-Y. Girard. The system F of variable types : fifteen years later. *Theoretical Computer Science* 45:159–192, 1986.

[8] C. Gunter, D. Scott. Semantic domains. In *"Handbook on Theoretical Computer Science"*, J. van Leeuwen, ed., North Holland, 1990, pp. 633–674.

[9] M. Gyssens, J. Paredaens, D. Van Gucht. A grammar-based approach towards unifying hierarchical databases. In *SIGMOD 89.*

[10] T. Imielinski, W. Lipski. Incomplete information in relational databases. *J. of ACM* 31(4):761–791, 1984.

[11] A. Jung. Personal communication. *June 1990.*

[12] A. Jung, L. Libkin, H. Puhlmann. Decomposition of domains. In *Proc. of the Conf. on Math. Foundations of Programming Semantics - 91*, to appear. Available as Technical Report MS-CIS-90-84, University of Pennsylvania, 1990.

[13] *Nested relations and Complex Objects in Databases* (S.Abiteboul, P.Fischer and H.-J.Schek eds.) *Springer LNCS*, Vol. 361, 1989.

[14] A. Ohori. A study on semantics, types and languages for databases and object-oriented programming. PhD Thesis, University of Pennsylvania, 1989.

[15] A. Ohori. Semantics of types for database objects. *2nd International Conference on Database Theory*, 1988.

[16] J. Paredaens, P. De Bra, M. Gyssens, D. Van Gucht. *The Structure of the Relational Datamodel.* Springer-Verlag, Berlin, 1989.

[17] M.A. Roth, H.F. Korth, A. Silberschatz. Extended algebra and calculus for nested relational databases. *ACM TODS*, 13(4):389–417, 1988.

[18] M.A. Roth, H.F. Korth, A. Silberschatz. Null values in nested relational databases. *Acta Informatica*, 26(7):615–642, 1989.

[19] V.N. Salii. *Lattices with Unique Complements.* AMS, Providence, RI, 1988.

[20] H.-J. Schek, M. Scholl. The relational model with relation-valued attributes. *Inform. Systems*, 11(2):137–147, 1986.

[21] D.S. Scott. Domains for denotational semantics. In *ICALP*, July 1982.

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

[23] K. Tanaka, T.-S. Chang. On natural join in object-oriented databases. In : *Proc. of Int. Conf. on Deductive and Object-Oriented Databases*, Kyoto, December 1989.

[24] J.D. Ullman. *Principles of Database Systems.* Pittman, 2nd ed., 1982.

[25] C. Zaniolo. Database relations with null values. *Journal of Computer and System Sciences* 28(1):142–166, 1984.