A Semantics-based Approach to Design of Query Languages for Partial Information

Leonid Libkin

Bell Laboratories
600 Mountain Ave., Murray Hill, NJ 07974, USA
email: libkin@bell-labs.com

Abstract. Most of work on partial information in databases asks which operations of standard languages, like relational algebra, can still be performed correctly in the presence of nulls. In this paper a different point of view is advocated. We believe that the semantics of partiality must be clearly understood and it should give us new design principles for languages for databases with partial information.

There are different sources of partial information, such as missing information and conflicts that occur when different databases are merged. In this paper, we develop a common semantic framework for them which can be applied in a context more general than the flat relational model. This ordered semantics, which is based on ideas used in the semantics of programming languages, cleanly intergrates all kinds of partial information and serves as a tool to establish connections between them.

Analyzing properties of semantic domains of types suitable for representing partial information, we come up with operations that are naturally associated with those types, and we organize programming syntax around these operations. We show how the languages that we obtain can be used to ask typical queries about incomplete information in relational databases, and how they can express some previously proposed languages. Finally, we discuss a few related topics such as mixing traditional constraints with partial information and extending semantics and languages to accommodate bags and recursive types.

1 Partial Information in Databases

Many aspects of database systems whose importance is evident in a variety of applications are yet to be adequately represented in practical database management systems. In many cases the reason for this is the lack of underlying theory. One of such problems is handling partial information in databases. While no one doubts that it must be dealt with, simply because in most applications we can not assume that the information stored in a database is perfect, the field has not been satisfactorily explored. Most results about partial information in databases are negative in their nature. They show what can *not* be done – efficiently or at all – if standard tools are used in the presence of partial information.

The main goal of this paper is to make a step toward a general theory of partial information. Partiality of information can be viewed as giving additional meaning to values that can be stored. Alternatively, one can regard it as constraining those values. Note, however, that such constraints are imposed on values that can be stored, and not on the whole database. Our goal is to represent these constraints in an adequately chosen mathematical framework, so that they can be reasoned about. Having found such a framework, we must demonstrate its usefulness. In this paper we concentrate on developing languages for partial information.

The main thesis of this paper is that, rather than showing what can not be done with standard tools, one should concentrate on designing new tools specifically for handling partial information. This thesis can be subdivided into two.

1. In order to understand partial information in databases, we have to know exactly what it means. That is, we have to have a *semantics* for partial information. We develop a formalism, whose roots can be found in [Bis81, BJO91, Gra91, IL84, JLP92, Lib91, Vas79], and whose main idea is that *partiality is represented via orderings on objects.*
2. We are not interested in semantics *per se*; the semantics that we define will help us find the right programming constructs for query languages for partial information. Our approach is based on [Car88, BBN91, BBW92, BLS$^+$94], and its gist is that operations *naturally* associated with datatypes should be used as the basis for the language design. The word "naturally" has a precise mathematical meaning, and it has to do with the properties of semantic domains of the datatypes used. Thus, we can formulate our second main principle, which says that *semantics suggests programming constructs.*

In the rest of this section we give a brief survey of the field of partial information in databases – to the extent we shall need it to motivate and substantiate our study.

1.1 Classical approach – null values

Name	Salary	Room	Telephone
John	15K	075	**ni**
Ann	17K	**ni**	**ni**
Mary	**ni**	351	x-1595

Name	Salary	Room	Telephone
John	15K	075	**ne**
Ann	17K	**un**	**ni**
Mary	**un**	351	x-1595

Fig. 1. Relations with nulls

Soon after Codd introduced his relational model, people realized that in real applications not all values may be present. For example, in the first relation in figure 1 that might be a part of a university or a corporation database, some values are missing and the symbol **ni** (no information) is used. Note that there could be several different reasons for using **ni**. This is reflected in the second relation in figure 1 where three kinds of nulls are used (cf. [LL86, RKS89, Zan84]). **ne** means nonexistent; that is, John does not have a phone. **un** means existing unknown; Mary is on payroll but the precise figure of her salary is unknown. And **ni** still means no information. For other kinds of nulls see [GZ88, LL93].

One of the most important achievements of the early work on partial information was an observation made in [Cod79]. Since every null value can be potentially replaced by a non-null value, each relation with nulls is represented by a *set* of relations without partial information. Moreover, this set could be considered as the *semantics* of the given incomplete relation. This idea was central to the seminal study [IL84] in which querying databases with nulls using standard languages like relational algebra was examined. The family of all complete relations that a relation R with nulls can represent was called a representation of R; we prefer the term semantics of R and denote it by $[\![R]\!]$. If q is a relational algebra query, we can ask q on $[\![R]\!]$, obtaining $q([\![R]\!]) = \{q(T) \mid T \in [\![R]\!]\}$. If we could find an relation R' such that $[\![R']\!] = q([\![R]\!])$, then we would be able to call R' the answer to q on R, that is, $q(R)$. However, for most classes of queries this is impossible. In fact, even milder definition of $q(R)$ leads to similar negative results.

Very little is known about null values in complex objects or nested relations, that is, relations whose attributes can be relation-valued themselves. An attempt to extend the results of [IL84] was made in [RKS89] (only for a restricted subclass of complex objects, those in partitioned normal form, cf. [AB86]), but later an error was found [LL91]. It was then shown [LL93] that some of the results can be recovered if equality of representations of incomplete complex objects is replaced by the Hoare equivalence, which will be defined later. However, [LL93] used the standard presentation of languages for complex objects, like in [TF86, SS86, Col90], and consequently inherited all of its problems and drawbacks. In particular, the description of the notion of null-extended join operator is almost one-page long, and many other operations are rather hard to grasp. The algebra for complex objects proposed in [Lib91] does not have adequate power to work with set-valued attributes. Thus, the problem of incorporating partial information into data models more complex than the standard (flat) relational model remains open.

1.2 Semantics of partiality

Order and partiality The key idea of our approach to semantics of partial information is that partiality is represented via orderings on objects. For the first time this idea appeared probably in [Vas79], and two years later it was further explored in [Bis81]. As a simple example, consider values that may occur in a database. Then **ni** is more partial, or less informative, than any total (nonpartial) value v such as 15K or 'Mary'. Therefore, we impose an order according to which $\mathbf{ni} \leq v$ for any total value v.

Most databases are obtained from base values by applying record and set constructors, so we need to extend the orderings respectively. For records the most natural way to do it is componentwise. For records with fields labeled by $l_1, \ldots, l_n$, we define

$$[l_1 : v_1, \ldots, l_n : v_n] \leq [l_1 : v_1', \ldots, l_n : v_n'] \quad \text{iff} \quad \forall i = 1, \ldots, n : \; v_i \leq v_i'$$

For sets there are various ways to extend a partial order, and typically the following one, perceived as a generalized subset ordering, $X \sqsubseteq Y$ iff $\forall x \in X \; \exists y \in Y : \; x \leq y$, was considered.

The idea of representing partiality via orders is central to our study. At this point, we would like to note that it can also be viewed as imposing certain constraints on *values* that can be stored, rather than the whole database, as is the case with most standard constraints. That is, **ni** and 123 are not just symbols; there is a certain semantic relationship between them, that is often not taken into account in the theory of partial information.

Constraints on null values Codd [Cod75] proposed to use three-valued logic to query databases with nulls, but serious problems with this approach were exemplified in [Gra77]. Instead, in [Gra77, Bis81] and a number of other papers it was suggested that one use Skolem variables to represent different occurrences of nulls. To represent various interconnections between those nulls, it was suggested to use constraints on the Skolem variables. For example, in the simplest case, called Codd tables, all Skolem variables are distinct. Inequality tables allow conditions like $x \neq y$ or $x \neq 4$ where x and y are variables. In conditioned tables, in addition to such constrains, a variable may occur more than once, and each tuple may have a constraint associated only with it.

In [AKG91] complexity of querying relational databases with incomplete information and constraints on nulls is studied thoroughly. A typical problem considered in [AKG91] is the following. Given a query q, a relation R with nulls and a set of constraints C, and a total relation (that is, relation without any occurrences of nulls) T, is it possible that one can find a relation T' in $[\![R]\!]$ such that T' satisfies all the constraints in C and $q(T') = T$. It was shown in [AKG91] that for many classes of constrained tables problems of this kind are very hard (i.e. NP, coNP or Π_2^{p}-complete), but in some restricted cases they are polynomial.

1.3 Semantics of collections

Assume we are given a collection of database objects with partial information. What is the semantics of such collection? It turns out that this question can not be answered unless we make certain assumptions about what kinds of collections can be supported. In what follows, we discuss three which are of particular importance for this paper: sets under closed and open world assumptions and disjunctive sets (or-sets). In section 4 we shall also consider bags.

R :

Name	Salary	Room
ni	**ni**	076
Mary	17K	**ni**

CWA → R_1

Name	Salary	Room
John	15K	076
Ann	**ni**	076
Mary	17K	561

OWA → R_2

Name	Salary	Room
John	15K	076
Ann	13K	325
Mary	17K	561

Fig. 2. Illustration to CWA and OWA

Open and closed worlds It was observed in [Rei78] that certain assumptions on the nature of partiality are to be made if we want to provide a notion of correctness of query evaluation algorithms. To explain these assumptions, consider relation R in figure 2. Once all or some information about missing values (**ni**'s) is known, we have a relation that represents better knowledge than R. However, there may be different assumptions about the values that are allowed in the new relation.

One possible interpretation, called the *closed world assumption* or *CWA*, states that we can only improve our knowledge about records that are already stored but can not invent new ones. For example, it is legal to add any record $\boxed{v_1}\boxed{v_2}\boxed{076}$ which improves upon the first record in R. It is also possible to add a record $\boxed{\text{Mary}}\boxed{\text{17K}}\boxed{561}$ which is better knowledge than that represented by the second record in R. However, it is *not* possible to add a record $\boxed{\text{Ann}}\boxed{\textbf{ni}}\boxed{561}$ as it does not improve any of the records already in the database. That is, the database is *closed* for adding new records.

Contrary to that, the *open world assumption* or *OWA* allows adding records to database as well as improving already existing records. Under the open world assumption, adding any record considered above to the database is perfectly legal. That is, the database is *open* for adding new records.

To summarize, Figure 2 shows how to replace missing values according to both assumptions. This interpretation of OWA and CWA is similar to the one typically used in databases with incomplete informations (cf. [IL84, Var86]) but slightly different from [Rei78] who used a logical setting. However, later we shall show that analogs of most of the results from [Rei78] hold in our setting as well.

D_1:

Name	SS#	Age
John	123456789	24
Mary	987654321	32

D_2:

Name	SS#	Age
John	123456789	27
Ann	564738291	25

merge $\longrightarrow$

Name	SS#	Age
John	123456789	$\langle 24, 27\rangle$
Mary	987654321	32
Ann	564738291	25

Fig. 3. Example of or-sets arising in merging databases

Disjunctive information The idea of using disjunctive information as a means to express partiality was already present in [Lip79, Lip81]. But it was not until almost ten years later that the first attempt was made to introduce disjunctions *explicitly* into the standard relational model. Consider the following example. Suppose we have two databases, D_1 and D_2 shown in figure 3. Assume that we merge D_1 and D_2. It is clear that records $\boxed{\text{Mary}}\boxed{987654321}\boxed{32}$ and $\boxed{\text{Ann}}\boxed{564738291}\boxed{25}$ should be in the resulting database. But what is the value of the Age field for John? Since SS# identifies people uniquely, we have *conflicting* information coming from two databases, and this conflict must be recorded in the newly created database until one finds out if John is 24 or 27 years of age. Therefore, both ages – 24 and 27 – are stored in the new database. However, the semantics of the Age attribute (which is now set-valued) is different from the usual interpretation of sets in databases. Rather than suggesting that John is both 24 and 27 years old, it says that John is 24 *or* 27.

Since such disjunctive sets, also called *or-sets*, have semantics that differs from the ordinary sets, we shall use a special notation $\langle\rangle$ for them. That is, in the result of merging D_1 and D_2, the value of the Age attribute for John is $\langle 24, 27\rangle$, see figure 3. While structurally just a set, it denotes an integer, which is either 24 or 27. That is, there are two different views of or-sets: structural, that concerns *representation*, and conceptual, that concerns *meaning*. This idea was present in the initial papers on or-sets [INV91a, INV91b] and later was formalized and worked out in [LW96].

1.4 Toward a general theory

There are a number of models for partial information in the database literature. Some of them are quite ad-hoc, based on specific needs arising in particular applications. We have seen two sources of partiality: null values and disjunctive information. (There are others; see, for example, [BDW91, Lib95a].) There are no solid theoretical foundations for any of these, nor are there any results that show how they are connected. Moreover, most models of partiality are developed only for the flat relational model, and virtually nothing is known for more complicated database models. This situation in the field of partial information was summarized in a recent survey [Kan90]:

> "... for the representation and querying of incomplete information databases, there are many partial solutions but no satisfactory full answer. It seems that the further away we move from the relational data model, the fewer analytical and algebraic tools are available."

Thus, to address the problem of partial information in databases and to move closer to satisfactory solutions that work for a large class of data models, one has to come up with new analytical tools and show their applicability not only in the study of the extended data models but also in the development of new query languages for databases with partial information. Making progress in this direction is the major motivation for this work. In this paper we develop a new approach to partial information that integrates all kinds of partiality within the same semantic framework. In addition to giving us necessary analytical and algebraic tools to study various kinds of partial information, this framework also naturally suggests operations that should be included into the language that works with partial information. Techniques that are developed for analyzing the structure of partial information can be applied to the study of the languages that deal with it.

Organization. In Section 2 we explore the first main principle of our approach saying that *partiality is represented via orders on objects.* First we briefly describe the main ideas of the approach of [BJO91, Lib91] that treats database objects as elements of certain partially ordered spaces of descriptions. Then we apply it in a typed setting, obtaining orderings for various kinds of collection type constructors. Thus, for the first time choosing orderings is tied with semantics of collections. Then we explain the difference between structural and conceptual representation of disjunctive information from the semantic point of view, and list some of the properties of semantic domains of collections which will be used for the language design.

In Section 3 we develop the second idea which says that *semantics suggests programming constructs.* We start by explaining the approach to the language design based on [Car88, BBN91, BBW92, BLS$^+$94] that suggests building languages for data around datatypes involved. Specifically, for each datatype constructor one needs introduction and elimination operations, and those can be obtained if one looks at the operations *naturally* – in the categorical sense – associated with the semantics of the datatypes. We show how to apply this approach to languages with partial information, and disjunctive information in particular. As two examples of applicability of obtained languages, we show that the algebra of [Zan84] can be viewed as a sublanguage of our language for sets, and we show how this language can be used to query equational tables, in which equality constraints are imposed on null values.

Finally, in Section 4, we discuss topics that should be further explored, but with some initial results already obtained. These include mixing traditional database constraints with partial information; recursive types and values in the presence of incompleteness of information; and extending our approach to bags (multisets).

All proofs can be found in [Lib94] (which is available by ftp). In those cases when proofs can also be found in journal or conference proceedings articles, additional references are given.

2 Semantics of Partial Information

The purpose of this section is to study the semantics of partial data. The unifying theme for various kinds of partial information is using ordered sets as their semantics, where the meaning of the order is "being more informative". Once orderings on values come into play, there is a need in new basic models for incomplete databases. We first describe an approach suggested in [BJO91] and further developed in [JLP92, Lib91, LL90] that, in a very general way, treats database objects as elements of certain ordered sets. Then we adapt this approach to the typed setting. For that we need to choose orderings on various kinds of collections. To do so, we formalize elementary updates on collections which improve our knowledge about the real world situation represented by that data, that is, add information. Then we characterize transitive closures of those updates, thus obtaining the orderings. We carry out this program for OWA and CWA sets and or-sets. We use the orderings to define the semantics of collections of partial objects. It will be shown that the semantics and the orderings agree naturally. We study important properties of semantic domains of partial data which will later be used to organize programming syntax.

2.1 Partial information and orderings on objects

It was discovered in [BJO91] that a representation of the underlying principles of relational database theory can be found in the theory of domains which has been developed as the basis of the denotational semantics of programming languages.

A database is a collection of descriptions, and the meaning $[\![d]\!]$ of a description d is the set of all possible objects described by it. Therefore, we can order descriptions by saying that a description d_1 is better than a description d_2 if it describes fewer objects, i.e. if it is a more precise description. For example, let d_1 and d_2 be the records in a relational database: $d_1 = [\text{Dept: CIS, Office: 176}]$, and $d_2 = [\text{Name: John, Dept: CIS, Office: 176}]$. If name, department and office are the only attributes, 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 $[\![d]\!]_2 \subseteq [\![d]\!]_1$.

If all descriptions of objects come from the same domain A which is partially ordered by $\leq$, then we define $[\![d]\!] \stackrel{\text{def}}{=} \{d' \in D \mid d' \geq d\} = \uparrow d$. Then $d_1 \leq d_2$ iff $[\![d_2]\!] \subseteq [\![d_1]\!]$. Sometimes it is helpful to restrict domains A to those in which every element $x \in A$ is bounded above by a maximal element $x_m \geq x$. The collection of maximal elements is denoted by $A^{\max}$, and the new semantic function then is $[\![x]\!]_{\max} = [\![x]\!] \cap A^{\max}$. This semantic function was used in [AKG91, Gra91, IL84].

Consistency in posets is another useful notion. Two elements $x, y \in A$ are called *consistent* if there exists $z \in A$ such that $x, y \leq z$. In the case of records this means *joinable* as in [Zan84] (i.e. they do not contradict each other): for example, $[\text{Name: John, Dept: } \mathbf{ni}, \text{Office: 176}]$ and $[\text{Name: John, Dept: CIS, Office: } \mathbf{ni}]$ are consistent as both of them are below d_2.

Note that if both d_1 and d_2 in our example above are stored in a relational database, then d_1 could be removed as it does not add any information. Generally, in the usual set interpretation of databases, if $x \leq y$, then x could be removed. Removing redundant elements leaves us with a collection of incomparable elements. Such collections are called *antichains*. That is, a subset X of an ordered set A is an antichain if $x \not< y$ for any $x, y \in A$.

The main idea of [BJO91] was that database objects are represented as antichains in domains, which are special kinds of posets used in semantics of programming languages. This was later refined in [Lib94] by requiring that database objects be antichains of compact elements; we shall return to this distinction later when we discuss recursive types. The approach has proved very fruitful. The concept of scheme was introduced in such a generalized setting, relational algebra operators were reconstructed, and functional and multivalued dependencies were defined and shown to possess the expected properties, see [BJO91, JLP92, Lib91].

However, this approach is too general, and we would like to adapt it to a typed setting. Complex objects. or nested relations, are constructed from values of base types (such as integers, strings etc.) by applying the record and the set type constructor. That is, their *types* are given by the following grammar:

$$t ::= b \mid [l_1 : t, \ldots, l_n : t] \mid \{t\}$$

where b ranges over a collection of base types, $[l_1 : t_1, \ldots, l_n : t_1]$ is the record type whose instances are records with fields l_is such that the value of the l_i field has type t_i, and $\{t\}$ is the set type constructor whose values are (for now) finite sets of values of type t.

Therefore, to obtain orderings for complex objects, we need to order base objects, records and sets. Orderings on base values are determined by null values that a given datatype allows. For example, in the case of three nulls **ne**, **ni** and **un** allowed for the type of naturals, the ordering is shown in figure 4. As was mentioned already, records are ordered componentwise. However, there is no "universal" way of ordering sets. The purpose of the next section is to identify some ways of doing it and associate them with various kinds of collections.

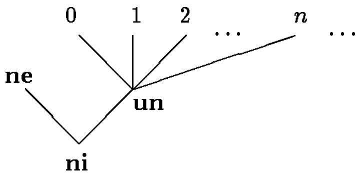

Fig. 4. Order on null values

2.2 Orderings on collections

Our general problem is the following. Given a poset $\langle A, \leq \rangle$ and the family of all collections (sets, or-sets etc.) over A, how do we order those? As usual, our interpretation of the partial order is "being more informative". What does it mean to say that one collection of partial descriptions is more informative than another? As two examples of families of collections over A that we would like to order, we consider $\mathbb{A}_{\text{fin}}(A)$, the family of finite antichains of A, and $\mathbb{P}_{\text{fin}}(A)$, the family of finite subsets of A.

A similar problem arises in the semantics of programming languages, most notably in the semantics of concurrency, cf. [Gun92]. Three orderings, called the *Hoare*, the *Smyth* and the *Plotkin* ordering have been proposed ([Gun92, Smy78, Plo76]):

(Hoare) $$X \sqsubseteq^{\flat} B \Leftrightarrow \forall x \in X \exists y \in Y \,:\, x \leq y$$

(Smyth) $$X \sqsubseteq^{\sharp} Y \Leftrightarrow \forall y \in Y \exists x \in Y \,:\, x \leq y$$

(Plotkin) $$X \sqsubseteq^{\natural} Y \Leftrightarrow X \sqsubseteq^{\flat} Y \text{ and } X \sqsubseteq^{\sharp} Y$$

All of them have been used for databases with partial information: the Hoare ordering in [Bis81, IL84, Lib91], the Smyth ordering in [BJO91, Oho90], the

Plotkin ordering in [PS93]. However, none of these papers addressed the question whether the chosen ordering is appropriate for the intended semantics of collections. Choosing the right orderings is the main purpose of this subsection. Our main claims are summarized in the table below.

Kind of collection	Ordering
Sets under CWA	Plotkin ($\sqsubseteq^{\natural}$)
Sets under OWA	Hoare ($\sqsubseteq^{\flat}$)
Or-sets	Smyth ($\sqsubseteq^{\sharp}$)

The technique we use to justify these claims is the following. We define "elementary updates" that add information. For example, for CWA databases such updates should add information to individual records. For OWA we may have additional updates that add records to a database. For or-sets, reducing the number of possibilities adds information as an or-sets denotes one of its elements. We formalize those updates and then look at their transitive closure. That is, a collection C_1 is more informative than C_2 if C_1 can be reached from C_2 by a sequence of elementary updates that add information. There are two ways to perform updates that add information, because redundancies represented by comparable elements could be removed. That is, one way is to keep all elements, even those that are comparable, and the other way is to remove redundancies, that is, to make sure that the result of each elementary update is an antichain again. These two ways lead to some orderings on either antichains of ordered sets or arbitrary subsets thereof. We shall consider both and show that they coincide.

Ordering CWA databases. In a closed world database, it is possible to update individual records but it is impossible to add new records. To understand what the elementary updates are, recall the example in figure 2. We view R_1 as more informative than R under the CWA. There could be more than one person in 076. That is, an incomplete record can be updated in various ways that give rise to a number of new records, and this is consistent with the closed world assumption, and this is how the first two records in R_1 are obtained. The third record in R_1 is obtained from the second record in R by adding the salary value. Thus, we see that the way the closed world databases are made more informative is by getting more information about individual records. The first picture in figure 5 illustrates those updates. We simply remove an element (record) from a database and replace it by a number of more informative elements (records).

There are two ways to formalize those updates, depending on whether arbitrary sets or only antichains are allowed. Let $X \subseteq A$ be a finite subset of the poset A. Let $x \in X$ and $X' \subseteq A$ be a finite nonempty subset of A such that $x \leq x'$ for all $x' \in X'$. Then we allow the following update:

$$X \overset{\text{CWA}}{\longmapsto} (X - x) \cup X'$$

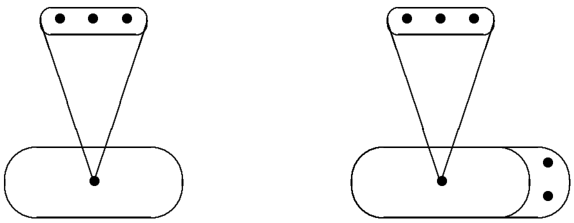

Fig. 5. Updates for CWA and OWA

For antichains, we need to impose two additional restrictions. First, X' must be an antichain, and second, the result must be an antichain. To ensure the second requirement is satisfied, we keep only maximal elements. That is, in the case of antichains the legitimate updates are

$$X \stackrel{\text{CWA}}{\longmapsto}_a \max((X - x) \cup X')$$

We now say that $X \sqsubseteq^{\text{CWA}} Y$ if $X, Y \subseteq A$ and Y can be obtained from X by a sequence of updates $\stackrel{\text{CWA}}{\longmapsto}$, that is, $\sqsubseteq^{\text{CWA}}$ is the transitive closure of $\stackrel{\text{CWA}}{\longmapsto}$ on $\mathbb{P}_{\text{fin}}(A)$. Similarly, $X \sqsubseteq_a^{\text{CWA}} Y$ if X, Y are finite antichains of A and Y can be obtained from X by a sequence of updates $\stackrel{\text{CWA}}{\longmapsto}_a$, that is, $\sqsubseteq_a^{\text{CWA}}$ is the transitive closure of $\stackrel{\text{CWA}}{\longmapsto}_a$ on $\mathbb{A}_{\text{fin}}(A)$. To justify the claim that the closed world databases must be ordered by the Plotkin ordering, we prove the following.

Theorem 1. a) *Let $X, Y \in \mathbb{P}_{\text{fin}}(A)$. Then $X \sqsubseteq^{\text{CWA}} Y$ iff $X \sqsubseteq^{\natural} Y$.*
b) *Let $X, Y \in \mathbb{A}_{\text{fin}}(A)$. Then $X \sqsubseteq_a^{\text{CWA}} Y$ iff $X \sqsubseteq^{\natural} Y$.* $\square$

Corollary 2. *Let X and Y be finite antichains in A such that $X \sqsubseteq^{\natural} Y$. Then it is possible to find a sequence of antichains $X_1, \ldots, X_n$ such that $X_1, \ldots, X_n \subseteq X \cup Y$ and $X \stackrel{\text{CWA}}{\longmapsto}_a X_1 \stackrel{\text{CWA}}{\longmapsto}_a \ldots \stackrel{\text{CWA}}{\longmapsto}_a X_n \stackrel{\text{CWA}}{\longmapsto}_a Y$.* $\square$

Ordering OWA databases. In an open world database, it is possible to update individual records and add new records. As in the case of the CWA databases, consider a simple example of relations R and R_2 in figure 2. Some of the records in R_2, that we view as a more informative one, are obtained by modifying records of R. However, one record, $\boxed{\text{Ann}}\boxed{\perp}\boxed{325}$ can not be obtained by modifying any record in R. The reason it was put there is that the database is open for new records. Under this interpretation, we view adding records as an update that adds information. In the above example, adding that record improves our knowledge about what can be a university or a company database of employees. This is illustrated by the second picture in figure 5. Not only do we allow replacing a record by a number of more informative records, but we also allow adding new records.

Similarly to the CWA case, there are two ways to formalize these updates, depending on whether arbitrary sets or only antichains are allowed. Let $X \subseteq A$ be a finite nonempty subset of the poset A. Let $x \in X$ and $X' \subseteq A$ be a finite subset of A such that $x \leq x'$ for all $x' \in X'$. Let X'' be an arbitrary finite subset of A. Then we allow the following updates:

$$X \overset{\text{OWA}}{\longmapsto} (X - x) \cup X' \quad \text{and} \quad X \overset{\text{OWA}}{\longmapsto} X \cup X''$$

For antichains, we impose an additional restriction that the result always be an antichain. We do it by keeping only maximal elements in the results. Therefore, in the case of antichains the legitimate updates are

$$X \overset{\text{OWA}}{\longmapsto}_a \max((X - x) \cup X') \quad \text{and} \quad X \overset{\text{OWA}}{\longmapsto} \max(X \cup X'')$$

We say that $X \sqsubseteq^{\text{OWA}} Y$ if $X, Y \subseteq A$ and Y can be obtained from X by a sequence of updates $\overset{\text{OWA}}{\longmapsto}$, that is, $\sqsubseteq^{\text{OWA}}$ is the transitive closure of $\overset{\text{OWA}}{\longmapsto}$ on $\mathbb{P}_{\text{fin}}(A)$. Similarly, $X \sqsubseteq_a^{\text{OWA}} Y$ if X, Y are finite antichains of A and Y can be obtained from X by a sequence of updates $\overset{\text{OWA}}{\longmapsto}_a$, that is, $\sqsubseteq_a^{\text{OWA}}$ is the transitive closure of $\overset{\text{OWA}}{\longmapsto}_a$ on $\mathbb{A}_{\text{fin}}(A)$. To justify the claim that the OWA databases must be ordered by the Hoare ordering, we prove

Theorem 3. a) *Let $X, Y \in \mathbb{P}_{\text{fin}}(A)$. Then $X \sqsubseteq^{\text{OWA}} Y$ iff $X \sqsubseteq^{\flat} Y$.*
b) *Let $X, Y \in \mathbb{A}_{\text{fin}}(A)$. Then $X \sqsubseteq_a^{\text{OWA}} Y$ iff $X \sqsubseteq^{\flat} Y$.* □

Corollary 4. *Let X and Y be finite antichains in A such that $X \sqsubseteq^{\flat} Y$. Then it is possible to find a sequence of antichains $X_1, \ldots, X_n$ such that $X_1, \ldots, X_n \subseteq X \cup Y$ and $X \overset{\text{OWA}}{\longmapsto}_a X_1 \overset{\text{OWA}}{\longmapsto}_a \ldots \overset{\text{OWA}}{\longmapsto}_a X_n \overset{\text{OWA}}{\longmapsto}_a Y$.* □

Ordering or-sets. We now define update rules for or-sets. We start with a simple example.

X_1 : $\langle$

Name	Salary	Room
John	$\perp$	076
Ann	$\perp$	$\perp$
Mary	17K	$\perp$

$\rangle$ $\xrightarrow{\text{or}-\text{set}}$ X_2 : $\langle$

Name	Salary	Room
John	$\perp$	076
Ann	13K	$\perp$

$\rangle$

There are two reasons why we view X_2 as a more informative or-set than X_1. First, additional information about Ann was obtained. It is now known that her salary is 13K. Second, one of the records was removed. Note that removing an element from an or-set makes it more informative. Indeed, while $\langle 1, 2, 3 \rangle$ is an integer which is either 1 or 2 or 3, $\langle 1, 2\rangle$ is an integer which is 1 or 2, so we have additional information that it can not be 3.

Therefore, we consider two types of updates on or-sets: improving information about individual records and removing elements:

$$X \stackrel{\text{or}}{\longmapsto} (X - x) \cup X' \qquad \text{if } x \in X \text{ and } x \leq x' \text{ for all } x' \in X' \text{ and } X' \neq \emptyset$$

$$X \stackrel{\text{or}}{\longmapsto} X - x \qquad \text{if } x \in X \text{ and } X - x \neq \emptyset$$

To redefine these updates for antichains, we must decide how redundancies in or-sets are removed. We suggest that only minimal elements be kept in the results. To see why, consider the following or-set with two comparable records:

$\langle$

Name	Room
John	076
John	**un**

$\rangle$

This or-set denotes a person whose name is John and who is either in room 076 or in an unknown room. The semantics of this is exactly as having one record for John in an unknown room. (This will be made precise in the next section.) Hence, we retain the minimal elements. Then the updates for antichains become

$$X \stackrel{\text{or}}{\longmapsto} \min((X - x) \cup X') \qquad \text{if } x \in X \text{ and } x \leq x' \text{ for all } x' \in X' \text{ and } X' \neq \emptyset$$

$$X \stackrel{\text{or}}{\longmapsto} X - x \qquad \text{if } x \in X \text{ and } X - x \neq \emptyset$$

Define $\sqsubseteq^{\text{or}}$ and $\sqsubseteq^{\text{or}}_a$ as the transitive closure of $\stackrel{\text{or}}{\longmapsto}$ and $\stackrel{\text{or}}{\longmapsto}_a$ respectively. To justify the last claim that the or-sets must be ordered by the Smyth ordering, we prove the following.

Theorem 5. a) *Let $X, Y \in \mathbb{P}_{\text{fin}}(A)$, $X, Y \neq \emptyset$. Then $X \sqsubseteq^{\text{or}} Y$ iff $X \sqsubseteq^{\sharp} Y$.*
b) *Let $X, Y \in \mathbb{A}_{\text{fin}}(A)$, $X, Y \neq \emptyset$. Then $X \sqsubseteq^{\text{or}}_a Y$ iff $X \sqsubseteq^{\sharp} Y$.* □

Corollary 6. *Let X and Y be finite antichains in A such that $X \sqsubseteq^{\sharp} Y$. Then it is possible to find a sequence of antichains $X_1, \ldots, X_n$ such that $X_1, \ldots, X_n \subseteq X \cup Y$ and $X \stackrel{\text{or}}{\longmapsto}_a X_1 \stackrel{\text{or}}{\longmapsto}_a \ldots \stackrel{\text{or}}{\longmapsto}_a X_n \stackrel{\text{or}}{\longmapsto}_a Y$.* □

2.3 Semantics of collections

We will need some notation. Recall that the family of finite antichains of a poset A is denoted by $\mathbb{A}_{\text{fin}}(A)$. By $\mathcal{P}^{\flat}(A)$ we mean the poset $\langle \mathbb{A}_{\text{fin}}(A), \sqsubseteq^{\flat}\rangle$, and by $\mathcal{P}^\sharp(A)$ we denote $\langle \mathbb{A}_{\text{fin}}(A), \sqsubseteq^\sharp \rangle$. These two constructions are the bases for the Hoare and the Smyth powerdomains used in semantics of concurrency, see [Gun92]. Note that $\mathcal{P}^\flat(A)$ is a join-semilattice, where the join operation is given by $X \sqcup^\flat Y = \max(X \cup Y)$, and $\mathcal{P}^\sharp(A)$ is a meet-semilattice, where the meet operation is given by $X \sqcap^\sharp Y = \min(X \cup Y)$.

Recall that the semantics of a database object d, which is an element of an ordered set A, is defined as the set of all elements of A that it can possibly denote, that is, $[\![d]\!] = \uparrow d = \{d' \in A \mid d' \geq d\}$. Following this definition and the results of the previous section, we can define the semantics of sets under OWA and CWA. Assume that elements of sets are taken from a partially ordered set A. Then we define the semantic functions $[\![\cdot]\!]_{\text{set}}^{\text{OWA}}, [\![\cdot]\!]^{\text{OWA}}, [\![\cdot]\!]_{\text{set}}^{\text{CWA}}, [\![\cdot]\!]^{\text{CWA}}$ where index "set" stands for the set semantics (as opposed to the antichain semantics for which we do not use an index), as follows:

$$[\![X]\!]_{\text{set}}^{\text{OWA}} = \{Y \in \mathbb{P}_{\text{fin}}(A) \mid X \sqsubseteq^\flat Y\} \qquad [\![X]\!]^{\text{OWA}} = \{Y \in \mathbb{A}_{\text{fin}}(A) \mid X \sqsubseteq^\flat Y\}$$
$$[\![X]\!]_{\text{set}}^{\text{CWA}} = \{Y \in \mathbb{P}_{\text{fin}}(A) \mid X \sqsubseteq^\natural Y\} \qquad [\![X]\!]^{\text{CWA}} = \{Y \in \mathbb{A}_{\text{fin}}(A) \mid X \sqsubseteq^\natural Y\}$$

In what follows, we shall mostly consider the open world assumption. Hence, if no superscript is used, it is assumed that we deal with the OWA. That is, $[\![X]\!] = [\![X]\!]^{\text{OWA}}$ and $[\![X]\!]_{\text{set}} = [\![X]\!]_{\text{set}}^{\text{OWA}}$.

A number of useful properties of these functions are summarized in the following proposition.

Proposition 7. *1. If $X, Y \subseteq_{\text{fin}} A$, then $[\![Y]\!]_{\text{set}}^{\text{OWA}} \subseteq [\![X]\!]_{\text{set}}^{\text{OWA}}$ iff $X \sqsubseteq^{\text{OWA}} Y$ iff $X \sqsubseteq^\flat Y$.*

2. If $X, Y \in \mathbb{A}_{\text{fin}}(A)$, then $[\![Y]\!] \subseteq [\![X]\!]$ iff $X \sqsubseteq_a^{\text{OWA}} Y$ iff $X \sqsubseteq^\flat Y$.

3. If $X \subseteq_{\text{fin}} A$, then $[\![X]\!] = [\![\max X]\!]$ and $[\![X]\!]_{\text{set}}^{\text{OWA}} = [\![\max X]\!]_{\text{set}}^{\text{OWA}}$.

4. If $X, Y \subseteq_{\text{fin}} A$, then $[\![Y]\!]_{\text{set}}^{\text{CWA}} \subseteq [\![X]\!]_{\text{set}}^{\text{CWA}}$ iff $X \sqsubseteq^{\text{CWA}} Y$ iff $X \sqsubseteq^\natural Y$.

5. If $X \subseteq_{\text{fin}} A$, then $[\![X]\!]_{\text{set}}^{\text{CWA}} = [\![\max X \cup \min X]\!]_{\text{set}}^{\text{CWA}}$ and $[\![X]\!]^{\text{CWA}} = [\![\max X \cup \min X]\!]^{\text{CWA}}$. $\square$

Closed world databases were initially defined in the logical setting. In particular, [Rei78] defined a CWA answer to a query as a certain set of tuples without incomplete information. In our terminology, this corresponds to finding an answer to a query with respect to the $[\![]\!]_{\text{max}}^{\text{CWA}}$ semantic function. It was proved in [Rei78] that the CWA query evaluation distributes over union and intersection, and that whenever a database is consistent with the negations of the facts stored in it, the OWA and the CWA query evaluation algorithms produce the same result. It was also proved that the minimal CWA answers contain exactly one tuple.

The following proposition shows that analogs of these results hold in our setting. Note that to say that a database X is consistent with negation of any fact stored in it, is the same as to say that any $y \notin X$ is consistent with some $x \in X$. In other words, if every $z \in A$ lies under some $z_m \in A^{\max}$, then $X \sqsubseteq^{\sharp} A^{\max}$. Finally, a domain of n-ary relations with one kind of nulls is the product of n copies of an infinite flat domain. In view of this, the proposition below says that the results of [Rei78] are preserved, at least in the spirit.

Proposition 8. *Let A be a poset such that each element is under an element of $A^{\max}$. Then*
1) If A is a product of n copies of infinite flat domains and $Y \in [\![X_1 \cap X_2]\!]^{\text{CWA}}_{\max}$, then $Y = Y_1 \cap Y_2$ where $Y_1 \in [\![X_1]\!]^{\text{CWA}}_{\max}$ and $Y_2 \in [\![X_2]\!]^{\text{CWA}}_{\max}$.
2) For any poset A, $[\![X_1 \cup X_2]\!]^{\text{CWA}}_{\max} = \{Y_1 \cup Y_2 \mid Y_1 \in [\![X_1]\!]^{\text{CWA}}_{\max}, Y_2 \in [\![X_2]\!]^{\text{CWA}}_{\max}\}$.
3) If $X \sqsubseteq^{\sharp} A^{\max}$, then $[\![X]\!]^{\text{CWA}}_{\max} = [\![X]\!]^{\text{OWA}}_{\max}$.
4) If X is bounded above in A, then a minimal nonempty $Y \in [\![X]\!]^{\text{CWA}}_{\max}$ is a singleton. □

Or-sets can be treated at both structural and conceptual levels. At the structural level we just define $[\![X]\!]^{\text{or}} = \{Y \in \mathbb{P}_{\text{fin}}(A) \mid X \sqsubseteq^{\sharp} Y\}$ (or using $\mathbb{A}_{\text{fin}}(A)$ if we need an antichain semantics.) The following proposition is the counterpart of proposition 7 for or-sets.

Proposition 9. *1. If $X, Y \subseteq_{\text{fin}} A$, then $[\![Y]\!]^{\text{or}} \subseteq [\![X]\!]^{\text{or}}$ iff $X \sqsubseteq^{\text{or}} Y$ iff $X \sqsubseteq^{\sharp} Y$.*

2. If $X, Y \in \mathbb{A}_{\text{fin}}(A)$, then $[\![Y]\!]^{\text{or}} \subseteq [\![X]\!]^{\text{or}}$ iff $X \sqsubseteq^{\text{or}}_a Y$ iff $X \sqsubseteq^{\sharp} Y$.

3. If $X \subseteq_{\text{fin}} A$, then $[\![X]\!]^{\text{or}} = [\![\min X]\!]^{\text{or}}$. □

Note that propositions 7 and 9 justify using maximal elements to remove redundancies from sets under OWA and using minimal elements to remove redundancies from or-sets. For sets under CWA, it is necessary to retain both minimal and maximal elements; the elements which are strictly in between can be removed as the fifth item in proposition 7 suggests.

Semantics of types and typed objects. The semantic functions above could also be used to define the semantic domains of *types*. For simplicity, assume that we have the following type system:

$$t ::= b \mid t \times t \mid \{t\} \mid \langle t \rangle$$

and that we are dealing with the open world assumption. Notice that we use pairs instead of records. Pairs are sufficient to simulate records and are easier to work with as notation does not become too complicated. We now define the *structural semantics* $[\![]\!]_s$ that corresponds to the structural interpretation of or-sets.

Suppose that for each base type b its semantic domain $[\![b]\!]_s$ is given. We define the semantic domains of all types inductively. Suppose we want to deal with antichains. Then

- $[\![t \times s]\!]_s = [\![t]\!]_s \times [\![s]\!]_s$.
- $[\![\{t\}]\!]_s = \langle \mathbb{A}_{\text{fin}}([\![t]\!]_s), \sqsubseteq^\flat \rangle = \mathcal{P}^\flat([\![t]\!]_s)$.
- $[\![\langle t \rangle]\!]_s = \langle \mathbb{A}_{\text{fin}}([\![t]\!]_s), \sqsubseteq^\sharp \rangle = \mathcal{P}^\sharp([\![t]\!]_s)$.

The structural semantics of objects is defined inductively.

- For each base type b and an element x of this type, $[\![x]\!]_s = \uparrow x = \{x' \in [\![b]\!]_s \mid x' \geq x\}$.
- If $x = (x_1, x_2)$, then $[\![x]\!]_s = [\![x_1]\!]_s \times [\![x_2]\!]_s$.
- If X is of type $\{t\}$, then $[\![X]\!]_s = [\![X]\!]^{\text{OWA}}$.
- If X is of type $\langle t \rangle$, then $[\![X]\!]_s = [\![X]\!]^{\text{or}}$.

Note that the last clauses in the definitions of type and object semantics say that we have defined the *structural semantics* of or-sets. That is, we viewed or-sets as collections and not as single elements they could represent. Our next goal is to define the *conceptual* semantics $[\![]\!]_c$ of or-sets.

First, for base types both semantics coincide, i.e. $[\![b]\!]_c = [\![b]\!]_s$. For other type constructors $[\![]\!]_c$ is defined as follows. Note that there are two possibilities for the semantics of the set type constructor, but the definition of the semantics of objects will work with both of them.

- $[\![t \times s]\!]_c = [\![t]\!]_c \times [\![s]\!]_c$.
- $[\![\{t\}]\!]_c = \langle \mathbb{A}_{\text{fin}}([\![t]\!]), \sqsubseteq^\flat \rangle = \mathcal{P}^\flat([\![t]\!]_c)$ or $[\![\{t\}]\!]_c = \langle \mathbb{P}_{\text{fin}}([\![t]\!]_c), \sqsubseteq^\flat \rangle$.
- $[\![\langle t \rangle]\!]_c = [\![t]\!]_c$.

The last clause corresponds to the fact that conceptually an or-set is just one of its elements. Semantics of each object is now going to be a finitely generated filter $F = \uparrow\{f_1, \ldots, f_n\} = \uparrow f_1 \cup \ldots \cup \uparrow f_n$. Again, we define it inductively.

- For each base type b and an element x of this type, $[\![x]\!]_c = \uparrow x = \{x' \in [\![b]\!]_c \mid x' \geq x\}$.
- If $x = (x_1, x_2)$, then $[\![x]\!]_c = [\![x_1]\!]_c \times [\![x_2]\!]_c$.
- Let $X = \{x_1, \ldots, x_n\}$ be a set of type $\{t\}$. Then $[\![X]\!]_c = \{Y \mid \forall i = 1, \ldots, n : Y \cap [\![x_i]\!]_c \neq \emptyset\}$. Here Y is taken from $\mathbb{P}_{\text{fin}}([\![t]\!]_c)$ or $\mathbb{A}_{\text{fin}}([\![t]\!]_c)$ depending on the definition of the semantics of types.
- Let $X = \langle x_1, \ldots, x_n \rangle$ be an or-set of type $\langle t \rangle$. Then $[\![X]\!]_c = [\![x_1]\!]_c \cup \ldots \cup [\![x_n]\!]_c$.

Before we prove that this semantic function possesses the desired properties, let us make a few observations. First, the definition of the semantics of or-sets coincides with the intended semantics of or-sets: an or-set denotes one of its elements. Second, to understand the semantics of pairs and sets, consider two simple examples. Let $x_1 = \langle 1, 2 \rangle$, $x_2 = \langle 3, 4 \rangle$. Assume that there is no ordering involved. The semantics of x_1 is then a set $\{1, 2\}$ and the semantics of x_2 is $\{3, 4\}$.

Therefore, $[\![(x_1, x_2)]\!]_c = \{(1,3),(1,4),(2,3),(2,4)\}$. Now consider (x_1, x_2). It is a pair whose first component is 1 or 2 and whose second component is 3 or 4. Hence, it is one of the following pairs: $(1,3),(1,4),(2,3),(2,4)$. And this is exactly what the semantic function $[\![]\!]_c$ tells us. For semantics of sets, consider $X = \{x_1, x_2\} = \{\langle 1,2\rangle, \langle 3,4\rangle\}$. It is is a set that has at least two elements: one is 1 or 2, and the other is 3 or 4. Hence, it must contain one of the following sets (since we believe in OWA): $\{1,3\},\{1,4\},\{2,3\},\{2,4\}$. Now look at $[\![X]\!]_c$. A set Y belongs to $[\![X]\!]_c$ if $Y \cap [\![\langle 1,2\rangle]\!]_c = Y \cap \{1,2\} \neq \emptyset$ and $Y \cap [\![\langle 3,4\rangle]\!]_c = Y \cap \{3,4\} \neq \emptyset$ which happens if and only if Y contains one of the four sets above. This justifies our definition of the conceptual semantics of sets.

Now we can prove the following.

Theorem 10. *For every object x of type t, $[\![x]\!]_c$ is a finitely generated filter in $[\![t]\!]_c$. Furthermore, if x and y are of type t and $x \leq y$ in $[\![t]\!]_s$, then $[\![y]\!]_c \subseteq [\![x]\!]_c$.* □

Corollary 11. *If x and y are objects of the same type, then $[\![x]\!]_s = [\![y]\!]_s$ implies $[\![x]\!]_c = [\![y]\!]_c$.* □

The converse is not true: $\langle\langle 1,2\rangle, \langle 3\rangle\rangle$ and $\langle\langle 1\rangle, \langle 2\rangle, \langle 3\rangle\rangle$ are structurally different objects of type $\langle\langle int\rangle\rangle$, but $[\![\langle\langle 1,2\rangle,\langle 3\rangle\rangle]\!]_c = [\![\langle\langle 1\rangle,\langle 2\rangle,\langle 3\rangle\rangle]\!]_c = \{1,2,3\}$.

Relationship between CWA sets, OWA sets and or-sets. There is a naturally arising question: do we really need all three kinds of collections – OWA sets, CWA sets and or-sets? Can not we just represent some of them using the others? The answer to this question is that we do need all three kinds of collections and no such representations exist. First, let us see what could be a representation of, say, OWA sets with or-sets. It could be a procedure that, given a poset A and $X \in \mathbb{A}_{\text{fin}}(A)$, calculates $Y \in \mathbb{A}_{\text{fin}}(A)$ such that $Z \in [\![X]\!]$ iff $Z \in [\![Y]\!]^{\text{or}}$. The following proposition tells us that it is impossible to do so.

Proposition 12. *For every poset A which is not a chain and has at least two elements, there exists $X \in \mathbb{A}_{\text{fin}}(A)$ such that for no $Y \in \mathbb{A}_{\text{fin}}(A)$ the following holds: 1) $[\![X]\!] = [\![Y]\!]^{\text{or}}$; 2) $[\![X]\!]^{\text{or}} = [\![Y]\!]$; 3) $[\![X]\!] = [\![Y]\!]^{\text{CWA}}_{\text{set}}$; 4) $[\![X]\!]^{\text{CWA}}_{\text{set}} = [\![Y]\!]$; 5) $[\![X]\!]^{\text{or}} = [\![Y]\!]^{\text{CWA}}_{\text{set}}$; 6) $[\![X]\!]^{\text{CWA}}_{\text{set}} = [\![Y]\!]^{\text{or}}$.* □

2.4 Properties of semantic domains of types

We did not define the semantics of types and objects for nothing. Our goal is to use the semantics as a guideline for the language design. In this subsection we establish a number of useful properties of semantic domains of types which be used extensively in the next section.

Recall that the structural semantics of types $\{t\}$ and $\langle t\rangle$ was defined as $\mathcal{P}^\flat([\![t]\!])$ and $\mathcal{P}^\sharp([\![t]\!])$ respectively. Let $\eta : A \to \mathcal{P}^\flat(A)$ or $\mathcal{P}^\sharp(A)$ be the singleton function: $\eta(x) = \{x\}$. Then both $\mathcal{P}^\flat(\cdot)$ and $\mathcal{P}^\sharp(\cdot)$ have nice characterizations as follows.

Proposition 13. *Let A be a poset. Then $\langle \mathcal{P}^\flat(A), \sqcup^\flat, \emptyset\rangle$ ($\langle \mathcal{P}^\sharp(A), \sqcap^\sharp, \emptyset\rangle$) is the free join-semilattice with bottom (free meet-semilattice with top) generated by A. That is, for every join-semilattice with bottom $\langle S, \vee, \bot\rangle$ (meet-semilattice with top $\langle S, \wedge, \top\rangle$) and every monotone map $f : A \to S$, there exists a unique semilattice homomorphism $f^+ : \mathcal{P}^\flat(A) \to S$ ($f^+ : \mathcal{P}^\sharp(A) \to S$) that makes the first (second) diagram below commute.*

$$\begin{tikzcd} A \arrow[r, "\eta"] \arrow[dr, "f"'] & \langle \mathcal{P}^\flat(A), \sqcup^\flat, \emptyset\rangle \arrow[d, dashed, "\exists! f^+"] \\ & \langle S, \vee, \bot\rangle \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "\eta"] \arrow[dr, "f"'] & \langle \mathcal{P}^\sharp(A), \sqcap^\sharp, \emptyset\rangle \arrow[d, dashed, "\exists! f^+"] \\ & \langle S, \wedge, \top\rangle \end{tikzcd}$$

So far the only semantic distinction between or-sets and sets showed up in different orderings for those and in different interpretations for conceptual semantics. We have not yet seen any results suggesting how these may interact. This is important for a language design, so that we would be able to distinguish between sets and or-sets. A natural way to study the connection between sets and or-sets is to look at the semantic domains of iterated types, that is, $\{\langle t\rangle\}$ and $\langle\{t\}\rangle$, and see how they are related. In other words, one has to find out what the relationship between $\mathcal{P}^\flat(\mathcal{P}^\sharp(A))$ and $\mathcal{P}^\sharp(\mathcal{P}^\flat(A))$ is. Here we have the following useful fact.

Theorem 14. (see also [Lib92, LW96]) *Given a finite set of finite sets $\mathcal{X} = \{X_1, \ldots, X_n\}$ where $X_i = \{x^i_1, \ldots, x^i_{k_i}\}$, let $F_{\mathcal{X}}$ be the set of functions $f : \{1, ..., n\} \to \mathbb{N}$ such that for any i: $1 \le f(i) \le k_i$. If all X_i's are subsets of A, define two maps α_a and β_a as follows:*

$$\alpha_a(\mathcal{X}) = \min_{f \in F_{\mathcal{X}}} {}_{\sqsubseteq^\flat}(\max\{x^i_{f(i)} \mid i = 1, \ldots, n\})$$

$$\beta_a(\mathcal{X}) = \max_{f \in F_{\mathcal{X}}} {}_{\sqsubseteq^\sharp}(\min\{x^i_{f(i)} \mid i = 1, \ldots, n\})$$

Then for any poset A, α_a restricted to $\mathcal{P}^\flat(\mathcal{P}^\sharp(A))$ and β_a restricted to $\mathcal{P}^\sharp(\mathcal{P}^\flat(A))$ are mutually inverse isomorphisms between $\mathcal{P}^\flat(\mathcal{P}^\sharp(A))$ and $\mathcal{P}^\sharp(\mathcal{P}^\flat(A))$. $\square$

Now, let us see what α_a does if there is no order involved. In this case an input to α_a can be considered as a set of or-sets:

$$\mathcal{X} \quad = \quad \{\langle x^1_1, \ldots, x^1_{k_1}\rangle, \ldots, \langle x^n_1, \ldots, x^n_{k_n}\rangle\}$$

Assume all x_j^is are distinct. Then $\alpha_a(\mathcal{X})$ is the or-set of sets

$$\langle \{x^1_{f(1)}, \ldots, x^n_{f(n)}\} \mid f \in F_{\mathcal{X}} \rangle$$

That is, all possible choices encoded by or-sets are explicitly listed. We shall use α_a as a programming primitive extensively in the next section.

The iterated construction $\mathcal{P}^{\flat\natural}(A) = \mathcal{P}^{\flat}(\mathcal{P}^{\natural}(A)) \cong \mathcal{P}^{\natural}(\mathcal{P}^{\flat}(A))$ possesses the following important property. Both join and meet operations can be defined on $\mathcal{P}^{\flat}(\mathcal{P}^{\natural}(A))$ and supply it with the lattice structure: $\mathcal{X} \sqcup^{\flat} \mathcal{Y} = \max^{\natural}(\mathcal{X} \cup \mathcal{Y})$ where $\max^{\natural}$ is taking maximal elements with respect to $\sqsubseteq^{\natural}$, and $\mathcal{X} \sqcap^{\flat} \mathcal{Y} = \max^{\natural}\{X \sqcap^{\natural} Y \mid X \in \mathcal{X}, Y \in \mathcal{Y}\}$. Moreover, it can be described via a well-known mathematical construction. This description will prove useful later, when we discuss programming primitives for or-sets.

Theorem 15. *For an arbitrary poset A, $\mathcal{P}^{\flat\natural}(A)$ is the free distributive lattice with top and bottom generated by A.* □

This result is quite robust and holds when some changes are made in the definitions of $\mathcal{P}^{\flat}(\cdot)$ and $\mathcal{P}^{\natural}(\cdot)$. In particular, if $\mathcal{P}^{\flat}_{\neq\emptyset}(A)$ and $\mathcal{P}^{\natural}_{\neq\emptyset}$ are defined as $\mathcal{P}^{\flat}$ and $\mathcal{P}^{\natural}$ except that the empty antichain is not allowed and $\mathcal{P}^{\flat\natural}_{\neq\emptyset}$ and $\mathcal{P}^{\natural\flat}_{\neq\emptyset}$ are respective compositions of $\mathcal{P}^{\flat}_{\neq\emptyset}$ and $\mathcal{P}^{\natural}_{\neq\emptyset}$, then the following holds.

Corollary 16. *For an arbitrary poset A, $\mathcal{P}^{\flat\natural}_{\neq\emptyset}(A)$ and $\mathcal{P}^{\natural\flat}_{\neq\emptyset}(A)$ are isomorphic. Moreover, $\mathcal{P}^{\flat\natural}_{\neq\emptyset}(A)$ is the free distributive lattice generated by A.* □

This fact is the key of the normalization process suggested in [LW96] as a means of incorporating conceptual semantics into the language. We shall come to it again later.

3 Languages for partial information

3.1 The Tannen-Cardelli thesis

In this subsection we give an overview of two principles of language design, which, when combined, provide a uniform way of organizing programming syntax around datatypes involved.

Suppose we want to design a language that works with objects given by some type system, like the one we had for complex objects. How do we choose primitives of such a language? The idea of Cardelli (see [Car88]) is that one should use *introduction* and *elimination* operations associated with type constructors as primitives of a programming language. The introduction operations are needed to construct objects of a given type whereas the elimination operations are used for doing computations over them. For example, record formation is the introduction operation for records, and projections are the elimination operations.

How does one find those introductions and elimination operations? Databases work with various kinds of collections. One approach (due to Tannen [BBW92, BTS91]) to finding the introduction and elimination operations for those collections is to look for operations *naturally* associated with them. To do so, one often characterizes the semantic domains of collection types via *universality properties*, which tell us what the introduction and the elimination operations are.

$$\begin{tikzcd} [\![t]\!] \arrow[r, "\eta"] \arrow[dr, "f"'] & \langle [\![\mathsf{C}(t)]\!], \Omega \rangle \arrow[d, "f^+"] \\ & \langle X, \Omega \rangle \end{tikzcd} \qquad \begin{tikzcd} [\![t]\!] \arrow[r, "\eta"] \arrow[dr, "f"'] & \langle [\![\mathsf{C}(t)]\!], \Omega \rangle \arrow[d, "ext(f)"] \\ & \langle [\![\mathsf{C}(s)]\!], \Omega \rangle \end{tikzcd}$$

Fig. 6. Operations naturally associated with collection types

Assume that we have a collection type constructor (like sets, bags, lists etc.) that we denote by $\mathsf{C}(\cdot)$. Then, for any type t, $\mathsf{C}(t)$ is the type of collections of elements of type t (e.g. sets or bags of type t). By *universality property* we mean that the following is true about $[\![\mathsf{C}(t)]\!]$, the semantic domain of type $\mathsf{C}(t)$. It is possible to find a set Ω of operations on $[\![\mathsf{C}(t)]\!]$ and a map $\eta : [\![t]\!] \to [\![\mathsf{C}(t)]\!]$ such that for any other Ω-algebra $\langle X, \Omega \rangle$ and a map $f : [\![t]\!] \to X$ there exists a unique Ω-homomorphism f^+ such that the first diagram in figure 6 commutes. If we are successful in identifying η and Ω, then we can make them the *introduction* operations. The reason is that now any object of type $\mathsf{C}(t)$ can be constructed from objects of type t by first embedding them into type $\mathsf{C}(t)$ by means of η, and then constructing more complex objects using the operations from Ω. The *elimination* operation is given by the universality property. That is, the general elimination operation is a higher-order operation that takes f as an input and returns f^+.

Combining these two ideas by Cardelli and Tannen gives us languages for many kinds of collections. Consider sets, assuming that the semantic domain of $\{t\}$ is the finite powerset of elements of t, that is, $\mathbb{P}_{\mathrm{fin}}([\![t]\!])$. For any set X, its finite powerset $\mathbb{P}_{\mathrm{fin}}(X)$ is the free semilattice generated by X. That is, the operations of Ω are $\emptyset$ and $\cup$ and η is the singleton formation: $\eta(x) = \{x\}$. Moreover, these operations can be applied for arbitrary types. That is, η is the *polymorphic* singleton; its type is $t \to \{t\}$ for any t. Similarly, $\cup$ is the polymorphic union of type $\{t\} \times \{t\} \to \{t\}$. Any set of type $\{t\}$ can be constructed from elements of type t using $\emptyset, \cup$ and η: $\{x_1, \ldots, x_n\} = \eta(x_1) \cup \ldots \cup \eta(x_n)$.

The operation that takes f into f^+ is the following

$$\begin{array}{lll} \mathit{fun}\ f^+[e,u](\emptyset) & = e \\ \quad |\ \ f^+[e,u](\{x\}) & = f(x) \\ \quad |\ \ f^+[e,u](A \cup B) & = u(f^+[e,u](A), f^+[e,u](B)) \end{array}$$

This operation f^+, often called *structural recursion* [BBN91, BBW92], depends on e and u which are interpretations of the operations of Ω on its range. Notice that if e and u do not supply the range of f^+ with the structure of a semilattice, then f^+ may not be well-defined. For example, if e is 0, f is the constant function that always returns 1, and u is $+$, then retaining duplicates may easily lead to a wrong cardinality function: $1 = f^+[0,+](\{1\}) = f^+[0,+](\{1,1\}) = 2$. To overcome this problem, one should require that e be interpreted as $\emptyset$ and u as $\cup$. Generally, the simplest way to ensure well-definedness is to require that $\langle X, \Omega\rangle$ be $\langle [\![\mathsf{C}(s)]\!], \Omega\rangle$ for some type s. Thus, we obtain the second diagram in figure 6.

The unique completing homomorphism is called $\mathit{ext}(f)$, the extension of f. Its semantics in the case of sets is $\mathit{ext}(f)\{x_1, \ldots, x_n\} = f(x_1) \cup \ldots \cup f(x_n)$. This justifies the name because $\mathit{ext}(f)$ "extends" f to sets. It is a polymorphic higher-order operation that takes f of type $t \to \{s\}$ and returns $\mathit{ext}(f) : \{t\} \to \{s\}$. This function is well-defined. Using *ext* together with η, $\emptyset$, $\cup$, projections and record formation, conditional and the equality test gives us precisely the nested relational algebra [BBW92] but the presentation is nicer than the standard ones, such as in [SS86, TF86]. Instead of *ext* one can use two functions: $\mathit{map}(f) : \{t\} \to \{s\}$ provided f is of type $t \to s$ (this function maps f over its input: $\mathit{map}(f)(\{x_1, \ldots, x_n\}) = \{f(x_1), \ldots, f(x_n)\}$) and $\mu : \{\{t\}\} \to \{t\}$ that flattens a set of sets: $\mu(\{X_1, \ldots, X_n\}) = X_1 \cup \ldots \cup X_n$. Diagrams in figure in 6 represent a well-known mathematical construction, which is going from an adjunction to the Kleisli category of its monad, and the fact that *ext* and *map* and μ are interchangeable follows from the general properties of the categorical notion of a monad, see [BW90].

This approach to the language design was shown to be extremely useful in the past few years, see [LW94a, LW94b, Suc94]. Here we apply it to partial information; the reader has probably already noticed the similarity between diagrams in figure 6 and proposition 13, which will give us the operations of the language.

3.2 Language for sets and its sublanguages

General operators and pairs

$$\frac{g : u \to s \quad f : s \to t}{f \circ g : u \to t} \qquad \frac{c : bool \;\; f : s \to t \;\; g : s \to t}{\mathit{if}\ c\ \mathit{then}\ f\ \mathit{else}\ g : s \to t} \qquad \frac{f : u \to s \quad g : u \to t}{(f, g) : u \to s \times t}$$

$$\overline{\pi_1^{s,t} : s \times t \to s} \qquad \overline{\pi_2^{s,t} : s \times t \to t} \qquad \overline{!^t : t \to unit}$$

$$\overline{Kc : unit \to Type(c)} \qquad \overline{id^t : t \to t} \qquad \overline{\leq_s : s \times s \to bool}$$

Set operators for partial information (given by $\mathcal{P}^\flat$)

$$\overline{\rho_2^{s,t} : s \times \{t\} \to \{s \times t\}} \qquad \overline{\eta^t : t \to \{t\}} \qquad \overline{\sqcup_t^\flat : \{t\} \times \{t\} \to \{t\}}$$

$$\overline{\mu_a^t : \{\{t\}\} \to \{t\}} \qquad \overline{empty^t : unit \to \{t\}} \qquad \frac{f : s \to t}{map_a(f) : \{s\} \to \{t\}}$$

Set operators without partial information (given by $\mathbb{P}_{\mathrm{fin}}$)

$$\overline{\rho_2^{s,t} : s \times \{t\} \to \{s \times t\}} \qquad \overline{\eta^t : t \to \{t\}} \qquad \overline{\cup_t : \{t\} \times \{t\} \to \{t\}}$$

$$\overline{\mu^t : \{\{t\}\} \to \{t\}} \qquad \overline{empty^t : unit \to \{t\}} \qquad \frac{f : s \to t}{map(f) : \{s\} \to \{t\}}$$

Fig. 7. Expressions of $\mathcal{NRL}$ and $\mathcal{NRL}_a$

Consider sets under the OWA. Since the semantic domain of type $\{t\}$ is $\mathcal{P}^\flat([\![t]\!])$, proposition 13 gives us the universality property and consequently introduction and elimination operations. Introduction operations are $\eta(x) = \{x\}$ and $X \sqcup^\flat Y = \max(X \cup Y)$, while the restricted form of elimination operation ext_a is given by $ext_a(f)(\{x_1, \ldots, x_n\}) = f(x_1) \sqcup^\flat \ldots \sqcup^\flat f(x_n) = \max(f(x_1) \cup \ldots \cup f(x_n))$. We prefer using the *map*-μ presentation. The semantics of those operations is given by $\mu_a(\{X_1, \ldots, X_n\}) = \max(X_1 \cup \ldots \cup X_n)$ and $map_a(f)(\{x_1, \ldots, x_n\}) = \max(\{f(x_1), \ldots, f(x_n)\})$. The index "a" stands for antichains.

If no order (partiality) is involved, then the semantics of $\{t\}$ is $\mathbb{P}_{\mathrm{fin}}([\![t]\!])$ which is the free join-semilattice with bottom generated by $[\![t]\!]$. Hence, the operations given by this universality property are the same as those for the language for OWA sets, except that max is not taken. For instance, $\cup$ is used instead of $\sqcup^\flat$.

The resulting language, $\mathcal{NRL}$ is precisely the nested relational algebra as has been mentioned (see [BBW92]).

Figure 7 contains expressions of two languages: $\mathcal{NRL}$ (nested relational language) of [BBW92] and $\mathcal{NRL}_a$ ($\mathcal{NRL}$ on antichains). Both languages share the general operators (the only exception is $\mathcal{NRL}$'s equality test instead of comparability test of $\mathcal{NRL}_a$). In the figure, we annotate expressions with their most general types. Since those types can be inferred, in what follows we shall omit them. $\mathcal{NRL}$ has all operations from the group of operations not dealing with partial information, and $\mathcal{NRL}_a$ has operations from the "set operations for partial information" group. Let us briefly recall the semantics of the operators that have not been explained already. $\leq_s$ is the comparability test at type s; that is, $\leq_s (x, y)$ evaluates to true if x, y are of type s and $x \leq y$ in $[\![s]\!]_s$. In other words,

- $(x, y) \leq_{s\times t} (x', y)' \Leftrightarrow x \leq_s x'$ and $y \leq_t y'$.
- $x \leq_{\{s\}} y \Leftrightarrow x \leq_s^\flat y$ (i.e. $\forall o \in x\ \exists o' \in y:\ o \leq_s o'$).

ρ_2 is the pair-with operation: $\rho_2(x, \{x_1, \ldots, x_n\}) = \{(x, x_1), \ldots, (x, x_n)\}$. *unit* is a special base type that has only one element. Its presence here is dictated by the fact that $\mathcal{NRL}$ is an algebra of *functions.* That is, to make a constant like $\emptyset$ into a function, we make it a function of type $unit \rightarrow \{t\}$ that always returns $\emptyset$. Composition of functions is denoted by $\circ$, pairing of functions is denoted by (f, g) and π_1 and π_2 are first and second projections.

Note that the languages are parameterized by an unspecified family of base types. That is, we view $\mathcal{NRL}$ and $\mathcal{NRL}_a$ as analog of relational algebra or calculus, which is the starting point for most languages for flat relations. Should one need additional types and operations on them (like real numbers and real arithmetic), they can be added easily. But the most important step in language design is to choose the operations that manipulate data, and this is what the operations of $\mathcal{NRL}$ and $\mathcal{NRL}_a$ are.

Now we are going to establish some properties of the languages. First, we do not need $\leq_s$ as a primitive at all types because it can be defined.

Proposition 17. *Assume that $\leq_b$ is given for any base type b. Then $\leq_s$ is definable in $\mathcal{NRL}_a$ without using $\leq_s$ as a primitive. Furthermore, under the assumption that $\leq_b$ can be tested in $O(1)$ time, the time complexity of verifying $x \leq_s y$ is $O(n^2)$, where n is the total size of x and y.* □

Using this, we can show that $\mathcal{NRL}$ is sufficient to simulate $\mathcal{NRL}_a$.

Theorem 18. *$\mathcal{NRL}_a$ is a sublanguage of $\mathcal{NRL}$ augmented with $\leq_b$ for all base types.* □

However, there is one subtle point. Assume that we have two sets X_1 and X_2 of type $\{t\}$ such that $\max X_1 = \max X_2$. That is, X_1 and X_2 represent the same object in $[\![\{t\}]\!]_s$. Let $f : \{t\} \to t'$ be a function definable in $\mathcal{NRL}$. Is it true that $f(X_1)$ and $f(X_2)$ represent the same object in $[\![t']\!]_s$? Unfortunately, the answer to this question is negative. To see why, consider x and y of type t such that $x \leq_t y$ and $x \neq y$. Assume that $g : t \to t'$ is such that $g(x)$ and $g(y)$ are not comparable by $\leq_{t'}$. Then $map(g)(\{y\}) = \{g(y)\}$ and $map(g)(\{x, y\}) = \{g(x), g(y)\}$. Even though $\max\{y\} = \max\{x, y\}$, we have $\max(map(g)(\{y\})) \neq \max(map(g)(\{x, y\}))$. The reason why this happens is that g is not a monotone function. Requiring monotonicity is sufficient to repair this problem. Define the following translation function $(\cdot)^\circ$ on objects that forces objects in the set-theoretic semantics into the objects in the antichain semantics:

- For x of base type b, $x^\circ = x$.
- For $x = (x_1, x_2)$, $x^\circ = (x_1^\circ, x_2^\circ)$.
- For $X = \{x_1, \ldots, x_n\}$, $X^\circ = \max\{x_1^\circ, \ldots, x_n^\circ\}$.

We say that a function $f : s \to t$ definable in $\mathcal{NRL}$ *agrees with the antichain semantics* if $x^\circ = y^\circ$ implies $f(x)^\circ = f(y)^\circ$. We say that it is *monotone* iff $x \leq_s y$ implies $f(x) \leq_t f(y)$.

Proposition 19. *A monotone function f definable in $\mathcal{NRL}$ agrees with the antichain semantics. If f is not monotone, then $map(f)$ does not agree with the antichain semantics.* □

Therefore, we would like to identify the subclass of monotone functions definable in $\mathcal{NRL}$. Unfortunately, it is not possible to do it algorithmically. Not being able to decide monotonicity is another reason why we prefer to view $\mathcal{NRL}_a$ as a sublanguage of $\mathcal{NRL}$ in which the antichain semantics can be modeled, rather than a separate language.

Theorem 20. *It is undecidable whether a function f definable in $\mathcal{NRL}$ is monotone.* □

There are some intersting anomalies of the antichain semantics. The most surprising of all is that $[\![\eta]\!]_s = [\![powerset]\!]_s$ or, in other words, $\mathcal{NRL}_a(powerset) = \mathcal{NRL}_a$. Indeed, since for any $Y \in \mathbb{P}_{\text{fin}}(X)$ we have $Y \subseteq X$ and hence $Y \sqsubseteq^\flat X$, then under the antichain semantics $[\![\mathbb{P}_{\text{fin}}(X)]\!]_s = [\![\max \mathbb{P}_{\text{fin}}(X)]\!]_s = [\![\{X\}]\!]_s = [\![\eta(X)]\!]_s$. There are two lessons we learn from this interesting collapse. First, as we have said already, it is better to view $\mathcal{NRL}_a$ as a sublanguage of $\mathcal{NRL}$ rather than a separate language. Second, *powerset* is *not* a good candidate to enrich expressiveness of the language. (Of course, the result of [SP94] which states that even very simple algorithms expressed with *powerset* need at least exponential space to be evaluated is a much stronger argument against *powerset*).

The next question we are going to address is that of conservativity of $\mathcal{NRL}$ over $\mathcal{NRL}_a$. Given a family of primitives $\mathbf{p}$ interpreted for both set theoretic and antichain semantics, we say that $\mathcal{NRL}(\leq_\flat, \mathbf{p})$[1] is *conservative* over $\mathcal{NRL}_a(\mathbf{p})$ if for any function f definable in $\mathcal{NRL}(\leq_\flat, \mathbf{p})$ and satisfying the condition that $f(x) = f(x)^\circ$ for any $x = x^\circ$, such f is definable in $\mathcal{NRL}_a(\mathbf{p})$. We do not know if $\mathcal{NRL}(\leq_\flat)$ is conservative over $\mathcal{NRL}_a$. However, we can show that it is conservative when augmented with aggregate functions. Instead of choosing a restricted set of aggregates, we use a general template suggested by [LW94a, LW94b]. This is the higher-order function $\sum(f)$ that takes a function $f : t \to \mathbb{N}$ and returns $\sum(f) : \{t\} \to \mathbb{N}$ given by $\sum(f)(\{x_1, \ldots, x_n\}) = f(x_1) + \ldots + f(x_n)$. Other operations on the type of naturals include multiplication and modified subtraction (monus) $\dot{-}$. The key idea in the proof of the proposition below is that using these additional functions we can encode objects using only natural numbers, cf. [LW94c].

Proposition 21. *$\mathcal{NRL}(\mathbb{N}, \sum, \cdot, \dot{-}, \leq_\flat)$ is conservative over $\mathcal{NRL}_a(\mathbb{N}, \sum, \cdot, \dot{-})$.* □

Example: Zaniolo's language In one of the first languages for partial information [Zan84] there is only one kind of nulls – **ni**. The ordering on records is defined component-wise and it is lifted to relations by using the Hoare ordering. Zaniolo's language was initially designed for flat relations only but here we show how to extend it to the nested relations. We shall use the notation $\downarrow x$ for $\{y \mid y \leq x\}$.

The main notion in the language is that of x-relation which is an equivalence class with respect to the Hoare ordering. That is, R_1 and R_2 are equivalent if $R_1 \sqsubseteq^\flat R_2$ and $R_2 \sqsubseteq^\flat R_1$. In our terminology this means that $\downarrow R_1 = \downarrow R_2$. Therefore, we can pick a canonical representative of each equivalence class: the canonical representative of the equivalence class of R is $\max R$. Clearly, $\downarrow R_1 = \downarrow R_2$ implies $\max R_1 = \max R_2$.

The next notion used for defining the operations in the language is that of generalized membership: $r \hat{\in} R$ iff $r \leq r'$ for some $r' \in R$. In other words, $r \hat{\in} R$ iff $r \in \downarrow R$ or $\{r\} \sqsubseteq^\flat R$. Using this notion, Zaniolo defined the following main operations:

$$R_1 \hat{\cup} R_2 = \max\{r \mid r \hat{\in} R_1 \text{ or } r \hat{\in} R_2\}$$

$$R_1 \hat{\cap} R_2 = \max\{r \mid r \hat{\in} R_1 \text{ and } r \hat{\in} R_2\}$$

$$R_1 \hat{-} R_2 = \max\{r \mid r \hat{\in} R_1 \text{ and } \neg(r \hat{\in} R_2)\}$$

Now we can see how operations are translated into the standard order-theoretic language we advocate in this paper:

$$R_1 \hat{\cup} R_2 \;=\; \max\{t \mid t \in \downarrow R_1 \text{ or } t \in \downarrow R_2\} \;=\; \max \downarrow R_1 \cup \downarrow R_2 \;=\; R_1 \sqcup^\flat R_2$$

[1] We use parenthesis to list types and operations added to the language.

$$R_1 \hat{\cap} R_2 \;=\; \max{\downarrow}R_1 \cap {\downarrow}R_2 \;=\; \max\{r_1 \wedge r_2 \mid r_1 \in R_1, r_2 \in R_2\} \;=\; R_1 \sqcap^\flat R_2$$

$$R_1 \hat{-} R_2 \;=\; \max\{t \mid t \in {\downarrow}R_1 \text{ and } \neg(t \in {\downarrow}R_2)\} \;=\; R_1 - {\downarrow}R_2$$

Thus, Zaniolo's union, intersection and difference are order-theoretic analogs of the usual set-theoretic union, intersection and difference. Next we notice that these operations are definable in $\mathcal{NRL}_a$ and hence in $\mathcal{NRL}$ augmented with orderings at base types. We have seen already that max is definable, so we only need the following lemma which is proved by an easy induction and definitions of $\sqcup^\flat$ and $\sqcap^\flat$.

Lemma 22. *If the least upper bound $\vee_b : b \times b \to b$ and the greatest lower bound $\wedge_b : b \times b \to b$ are given for any base type b, then the least upper bound $\vee_s : s \times s \to s$ and the greatest lower bound $\wedge_s : s \times s \to s$ are definable in $\mathcal{NRL}_a$ for every type s.* $\square$

The last operation of Zaniolo's language is the join (we omit projection and selection as these are standard and of course definable in $\mathcal{NRL}_a$). The join with respect to a set X of attributes was defined as

$$R_1 \bowtie_X R_2 \quad := \quad \max\{t_1 \vee t_2 \mid t_1 \hat{\in} R_1, \quad t_2 \hat{\in} R_2, \quad t_1 \text{ and } t_2 \text{ are total on } X\}$$

Without the condition that t_1 and t_2 must be total on X that translates into $\max\{t_1 \vee t_2 \mid t_1 \in R_1, t_2 \in R_2\}$ and hence is definable in $\mathcal{NRL}_a$ by taking cartesian product of R_1 and R_2 and mapping $\vee$ over it. In the case of flat relations, it is also possible to check if the value of a projection is **ni** since **ni** is available as a constant of base types now. Hence, the totality condition can be checked, and since selection is definable, so is $\bowtie_X$. Summing up, we have

Theorem 23. *The language of Zaniolo is a sublanguage of $\mathcal{NRL}_a$, and hence $\mathcal{NRL}$.* $\square$

Notice that in the case of model with one null **ni** we do not have to require orderings on base types as these are definable using just equality test.

3.3 Language for sets and or-sets

Proposition 13 gives us the properties of semantic domains of or-set types which are necessary to find the programming primitives. Notice that if no ordering is involved, then structurally or-sets and sets are indistinguishable. Hence, in this case all or-set operations are the same as in the case of sets, and we only add prefix *or* and change types $\{t\}$ to $\langle t \rangle$. In the case of ordered semantics, it is only the ordering and removal of redundancies that are different. Hence, we shall have analogs of all operations of the set language but the semantics is different: $or\_map_a(f)(\langle x_1, \ldots, x_n\rangle) = \min(\langle f(x_1), \ldots, f(x_n)\rangle)$, $or\_\mu_a(\langle X_1, \ldots, X_n\rangle) = \min(X_1 \cup \ldots \cup X_n)$ and $X \sqcap^\sharp Y = \min(X \cup Y)$.

So far there is no interaction of sets and or-sets present in the language. Since any operator providing such interaction must have source and target types involving both sets and or-sets, theorem 14 suggests what this operator could be. Its type is $\{\langle t\rangle\} \to \langle\{t\}\rangle$. For the ordered case, it is α_a of theorem 14. For unordered case, it is the following operator α:

$$\begin{aligned} &\alpha(\{\langle x^i_j \mid 1 \le j \le k_i\} \mid i = 1, \ldots, n\rangle) \\ &= \langle\{x^i_{f(i)} \mid i = 1, \ldots, n\} \mid f : \{1, \ldots, n\} \to \mathbb{N}, \forall i : 1 \le f(i) \le k_i\rangle \end{aligned}$$

(or, compactly, $\alpha(\mathcal{X}) = \langle\{x^i_{f(i)} \mid i = 1, \ldots, n\} \mid f \in \mathcal{F}_{\mathcal{X}}\rangle$ using the notation of theorem 14).

Since or-sets are ordered by the Smyth ordering and redundancies are removed by taking minimal elements, we augment the definitions of orderings on complex objects and forcing sets into antichains from the previous section as follows:

- $x \le_{\langle s\rangle} y \Leftrightarrow x \le^\sharp_s y$ (i.e. $\forall o' \in y\ \exists o \in x :\ o \le_s o'$)
- $\langle x_1, \ldots, x_n\rangle^\circ = \min\langle x^\circ_1, \ldots, x^\circ_n\rangle$

Definition. *The language or-$\mathcal{NRL}$ is defined as $\mathcal{NRL}$ augmented by the or-set constructs without ordering from figure 8 and α, see [LW96]. The language or-$\mathcal{NRL}_a$ is defined as $\mathcal{NRL}_a$ augmented by the or-set constructs for ordered domains from figure 8 and α_a.*

Some useful properties of or-$\mathcal{NRL}$ and or-$\mathcal{NRL}_a$ are summarized in the theorem below.

Theorem 24. *1. If $\le_b$ is given at any base type b, then $\le_s$ is definable in or-$\mathcal{NRL}_a$ without using $\le_s$ as a primitive.*

2. *Under the assumption that $\le_b$ can be tested in $O(1)$ time, the time complexity of verifying $x \le_s y$ is $O(n^2)$, where n is the total size of x and y.*
3. *or-$\mathcal{NRL}_a$ is a sublanguage of or-$\mathcal{NRL}(\le_b)$.*
4. *For any two objects x, y of type s, $x \le_s y$ iff $x^\circ \le_s y^\circ$.*
5. *For any operator g_a of or-$\mathcal{NRL}_a$ and the corresponding operator g of or-$\mathcal{NRL}$ the following holds: $g_a(x) = g(x)^\circ$ whenever x is a legitimate input to g_a (that is, $x = x^\circ$).*
6. *Any monotone function f definable in or-$\mathcal{NRL}$ agrees with the antichain semantics. If f is not monotone, then $map(f)$ and $or\_map(f)$ do not agree with the antichain semantics.*
7. *It is undecidable whether a function f definable in or-$\mathcal{NRL}$ is monotone.* □

Or-Set operartions without ordering

$$\frac{}{or\_\rho_2{}^{s,t} : s \times \langle t \rangle \to \langle s \times t \rangle} \qquad \frac{}{or\_\eta^t : t \to \langle t \rangle} \qquad \frac{}{or\_\cup^t : \langle t \rangle \times \langle t \rangle \to \langle t \rangle}$$

$$\frac{}{or\_\mu^t : \langle\langle t \rangle\rangle \to \langle t \rangle} \qquad \frac{}{or\_empty^t : unit \to \langle t \rangle} \qquad \frac{f : s \to t}{or\_map\ f : \langle s \rangle \to \langle t \rangle}$$

Or-Set operations for ordered domains (given by $\mathcal{P}^\natural$)

$$\frac{}{or\_\rho_2{}^{s,t} : s \times \langle t \rangle \to \langle s \times t \rangle} \qquad \frac{}{or\_\eta^t : t \to \langle t \rangle} \qquad \frac{}{\sqcap_t^\natural : \langle t \rangle \times \langle t \rangle \to \langle t \rangle}$$

$$\frac{}{or\_\mu_a^t : \langle\langle t \rangle\rangle \to \langle t \rangle} \qquad \frac{}{or\_empty^t : unit \to \langle t \rangle} \qquad \frac{f : s \to t}{or\_map_a\ f : \langle s \rangle \to \langle t \rangle}$$

Interaction of sets and or-sets without ordering

$$\frac{}{\alpha^t : \{\langle t \rangle\} \to \langle\{t\}\rangle}$$

Interaction of sets and or-sets for ordered domains

$$\frac{}{\alpha_a^t : \{\langle t \rangle\} \to \langle\{t\}\rangle}$$

Fig. 8. Expressions of *or*-$\mathcal{NRL}$ and *or*-$\mathcal{NRL}_a$

Not let us look at the conceptual semantics $[\![]\!]_c$ of the or-set operators of *or*-$\mathcal{NRL}$ and *or*-$\mathcal{NRL}_a$.

Theorem 25. *The following equations hold:*

1. $[\![or\_\mu_a(x)]\!]_c = [\![x]\!]_c$.
2. $[\![\alpha_a(x)]\!]_c = [\![x]\!]_c$.
3. $[\![or\_\rho_2(x)]\!]_c = [\![x]\!]_c$.

4. $[\![x \sqcap^\sharp y]\!]_c = [\![x]\!]_c \cup [\![y]\!]_c$.

5. $[\![or\_map_a(f)(\{x_1, \ldots, x_n\})]\!]_c = [\![f(x_1)]\!]_c \cup \ldots \cup [\![f(x_n)]\!]_c$.

Moreover, for or-$\mathcal{NRL}$ the same equations hold if finite powerset is used instead of $\mathcal{P}^\flat(\cdot)$ to give semantics of $\{t\}$. □

The intuition behind the first three equations is that or_μ, or_ρ_2 and α do not change the meaning. Indeed, consider $x = \langle\langle 1, 2\rangle, \langle 2, 3\rangle\rangle$. The meaning of x is an or-set which is either $\langle 1, 2\rangle$ or $\langle 2, 3\rangle$. Hence, x is an integer which is either 1 or 2 or 3. But this is the same as the meaning of $\langle 1, 2, 3\rangle = or\_\mu(x)$. For α, the meaning of $y = \{\langle 1, 2\rangle, \langle 3\rangle\}$ is a set whose first element is 1 or 2 and whose second element is 3. That is, y is either $\{1, 2\}$ or $\{2, 3\}$, and its meaning is the same as that of $\langle\{1, 2\}, \{2, 3\}\rangle = \alpha(y)$.

It was shown in [LW96] that if or_μ, α and or_ρ_2 are repeatedly applied to subobjects of an object x while possible, then a) the process will eventually terminate and b) the result of this process does not depend on the sequence in which those operations were applied to subobjects of x. (As explained in more detail in [Lib95b], it is important that duplicates obtained during this process be retained until the very last application.) The result uniquely determined by such a process is called a *normal form* and denoted by *normalize*(x). It can be seen that if x has or-sets in it, then the type of *normalize*(x) is $\langle t\rangle$ where t does not have any or-set brackets. The intuitive meaning of *normalize*(x) is listing all possibilities encoded by x. Of course this should not change the meaning. Now, with the help of theorem 25 we can formulate this precisely.

Corollary 26. $[\![normalize(x)]\!]_c = [\![x]\!]_c$. □

This corollary is formulated for the set theoretic semantics, because existence and well-definedness of *normalize* was proved only for the set semantics in [LW96]. Extending this result in various ways, including antichain semantics, is the subject of a separate paper [Lib95b].

Concluding this section, we give a simple example of applicability of or-$\mathcal{NRL}$ to classical problems of incomplete information in relational databases by showing how to use it to solve the membership problem for equational tables.

Example: Membership problem for equational tables in or-$\mathcal{NRL}$ Recall that equational tables are relations where variables can be used as well as total values, and each variable may occur more than once. The membership problem is to determine, given an equational table and a relation without variables, if the relation is a possible world for the table. That is, if it is possible to instantiate variables to values such that the table will be instantiated into the given relation.

It is known that this problem is $\mathcal{NP}$-complete [AKG91], so we can not hope to give a solution that does not use the expensive α.

For simplicity of exposition, assume that we have a base type b having both variables $x_1, \ldots$ and values $v_1, \ldots$ and that it is possible to distinguish between variables and values. A relation R is an object of type $\{b \times b\}$ such that no variable occurs in it. A table T is also an object of type $\{b \times b\}$ but now variables may occur. It is possible to find the set of all variables that occur in T using the fact that *select* is defiable in $\mathcal{NRL}$ (cf. [BBW92]):

$$\text{VAR}_T \quad := \quad \mathit{select}(\mathit{is\_variable}) \circ \mathit{map}(\pi_1)(T) \;\cup\; \mathit{select}(\mathit{is\_variable}) \circ \mathit{map}(\pi_2)(T)$$

All values that occur in R can be found as

$$\text{VAL}_R \quad := \quad \mathit{map}(\pi_1)(R) \;\cup\; \mathit{map}(\pi_2)(R)$$

In *or*-$\mathcal{NRL}$ it is possible to define $\mathit{powerset}_{\text{or}} : \{t\} \to \langle\{t\}\rangle$ which lists all subsets of a given set. This is done by first taking a set $\{x_1, \ldots, x_n\}$ and producing a new object $\{\langle\{x_1\}, \{\}\rangle, \ldots, \langle\{x_n\}, \{\}\rangle\}$ and then applying α to it and mapping μ over the result. So, the next step is to compute $\mathit{powerset}_{\text{or}}(\mathit{cartprod}(\text{VAR}_T \times \text{VAL}_R))$ and select those sets in it in which every variable from VAR_T occurs exactly once. We denote this resulting object of type $\langle\{b \times b\}\rangle$ by ASSIGN. Each element of ASSIGN can be viewed as an assignment of values to variables, so it can be applied to T in the following sense. For every x in ASSIGN (which is a set of pairs variable-value), we can write a function that substitutes each variable in T by the corresponding value, and then map this function over ASSIGN. The reader is invited to see how such a function can be written in *or*-$\mathcal{NRL}$.

The resulting object is now X of type $\langle\{b \times b\}\rangle$ which is the or-set of all possible relations that can be obtained from T by using valuation maps whose values are in VAL_R. Therefore, R is a possible world for T if and only if R is a member of X. To verify this, we write $\mathit{or\_map}(\lambda x.\mathit{eq}(x, R))(X)$ and then check if *true* is in the result. This gives us the membership test.

It is interesting to note that the membership problem for Codd tables, while being of polynomial time complexity, requires solving the bipartite matching problem which can be reformulated as a problem of finding a system of distinct representatives, see [AKG91]. Therefore, the power of $\mathcal{NRL}$ is too limited to solve the membership problem even for Codd tables, because the bipartite matching problem can not be solved in it [Lib94]. However, with the power of α, the language can solve a much more complicated membership for equational tables.

4 New directions

4.1 Traditional constraints and partial information

In this paper we developed type systems and languages for databases with partial information. The next important step will be to accommodate traditional database constraints into the model. Relatively little is known about constraints in relational databases with nulls (see [AM86, Gra91, PDGV89, Tha91, Tha89]) and virtually nothing is known about constraints for other kinds of partial information. To the best of our knowledge, no work has been done on understanding how the ordering interacts with constraints.

There are several possible approaches to the study of interaction of traditional constraints with partial information. Since we advocate the order-theoretic models of databases and consider rather complicated type systems, we believe one should try to apply the approach that formalizes constraints independently of the particular kind of data structures involved. For example, one may use the lattice theoretic approach to dependencies and normalization developed in [DLM92, Day92] or define dependencies as certain classes of first order formulae as in [Fag82].

Another useful idea is to introduce analogs of some constrains for databases with partial information in a "disjunctive" manner [AM86, Tha89]. Following [Tha89], we consider keys. In a usual relational database, a set K of attributes is a key if $\pi_K(t_1) \neq \pi_K(t_2)$ for any two distinct tuples t_1 and t_2. Suppose we have a relational database in which only one kind of nulls, **ni**, is allowed, and the order is given by $\mathbf{ni} \leq v$ for any v. Then a family $\mathcal{K} = \{K_1, \ldots, K_n\}$ of sets of attributes is called a *key set* [Tha89] if for any two distinct tuples t_1 and t_2, there exists a $K_i \in \mathcal{K}$ such that t_1 and t_2 are defined on K_i (that is, none of the K_i-values is **ni**) and $\pi_{K_i}(t_1) \neq \pi_{K_i}(t_2)$. For relations without null values this simply means that $\bigcup \mathcal{K}$ is a key. A key set if minimal if all K_is are singletons. The disjunctive nature of such constraints matches the usual key constrains in the closed world semantics.

Proposition 27. *For any relation R with* **ni** *null values and a set K of attributes, $\mathcal{K} = \{\{k\} \mid k \in K\}$ is a minimal key set iff $\pi_{K \cap \mathrm{def}(t,t')}(t) = \pi_{K \cap \mathrm{def}(t,t')}(t')$ implies $t = t'$, where $\mathrm{def}(t,t')$ is the set of attributes on which both t and t' are defined. Furthermore, this implies that for any $T \in [\![R]\!]^{\mathrm{CWA}}_{\max}$ with card $T \leq$ card R, K is a key of T.* □

The converse to the last statement is not true. Consider $R = \{(\mathbf{ni}, 1), (2, 1)\}$. Then for any T as in the statement of the proposition, the first attribute is a key, but it is not a key set for R.

We believe that this idea of making one constraint into a family while maintaining a close connection with the intended semantics can be quite productive. The concept of a key set can be reformulated as $\forall t, t' \; \forall K \in \mathcal{K} : (K \subseteq \mathrm{def}(t,t') \Rightarrow \pi_K(t) = \pi_K(t')) \;\; \Rightarrow \;\; t = t'$. This in turn implies that $\bigcup \mathcal{K}$ is a key for any $T \in [\![R]\!]^{\mathrm{CWA}}_{\max}$ and shows that keys can be further generalized to functional dependencies and probably a to greater class of dependencies given in a first order language with equality.

4.2 Recursive types and values

The discussion in this subsection assumes some knowledge of the formal semantics of programming languages. The complex object data model, which was the main object of study in this paper, usually serves as the underlying model for object-oriented databases. But object-oriented databases include more than that. In particular, they often deal with recursive values. In many models recursive values are represented by oids; in practice, these are implemented as pointers. However, the formal semantics of recursive types and values, and in particular recursive types and values in the presence of partial information, must be worked out.

Since semantics of recursive types is usually obtained as a limit construction, this suggests using domains instead of arbitrary posets. Assume that we add a recursive type constructor to the type system:

$$t := x \mid b \mid unit \mid t \times t \mid \{t\} \mid \mu x.t$$

where x ranges over type variables, and $\mu x.t$ is the recursive type constructor (x must be free in t). Since semantics of recursive types is usually obtained as a solution to an equation, which in turn is a (co)limit in some category, we have to switch to categories of domains from categories of posets. A *domain* is a poset in which every directed set has a least upper bound and compact elements form a basis. A compact element is characterized by the property that $c \leq \bigsqcup X$ implies $c \leq x$ for some $x \in X$. Compact elements form a basis of D if for every $x \in D$, $\mathbf{K}_x = \{c \mid c \leq x, c \text{ compact}\}$ is directed and $x = \bigsqcup \mathbf{K}_x$.

It was suggested in [Gun85] that one formulate a number of requirements on the category of domains in which the semantics of types is to be found. In [Gun85] such requirements were formulated for type systems suitable for traditional functional languages, but those do not use the set type constructor. Following [Gun85], let us try to formulate a number of requirements on the category of domains $\mathbf{C}$ in which a semantics of recursive complex object types can be found. First of all, its objects must be closed under $\times$ (product type) and $\wp^\flat(\cdot)$ which is $\mathsf{Idl}(\mathcal{P}^\flat(\mathbf{K}\cdot))$, the ideal completion of $\mathcal{P}^\flat(\mathbf{K}\cdot)$. Second, it must contain the domains of base types (which are usually flat domains or those similar to posets in figure 4). Third, equations of form $D = \mathbb{F}(D)$, where $\mathbb{F}$ is a functor composed from the constant base type functors, products and $\wp^\flat(\cdot)$, must have a solution in $\mathbf{C}$. This guarantees that the semantics of recursive types can still be found in $\mathbf{C}$.

Of course a number of categories satisfy these requirements, but most of them contain too many domains that never arise as domains of types. If we interpret compact elements as objects that can actually be stored in a database, then having an object x that can be stored and an object y that is less informative than x, we should be able to store y as well, provided or-sets are not used. That is, there is one additional condition saying that the compact elements must form an ideal, i.e. $\downarrow\mathbf{K}D = \mathbf{K}D$. Now we call a category of domains that satisfy all these conditions a *database category.*

Proposition 28. *The following are examples of database categories:*
1) $\mathbf{C}_1$*, the category of domains in which there is no infinite chain under any compact element.*
2) $\mathbf{C}_2$*, the category of domains in which the number of elements under any compact element is finite.*
3) Subcategories of $\mathbf{C}_1$ *and* $\mathbf{C}_2$ *in which all ideals are distributive lattices and/or maps are required to preserve compactness.*
4) The category of dI-domains and stable maps (see [Gun92] for the definition).
$\square$

So, we have a number of categories in which semantics of recursive complex object types can be found. But this is not the end of the story, because there are two major issues that must be addressed. First, these conditions are no longer satisfied if we add the or-set type constructor. Second, all recursive database objects have finite representation and could be stored in a database. But we can easily see that they are not necessarily compact elements in the domains of their types. For example, consider $\mu x.string \times x$. Its elements are infinite sequences of strings, and compact elements are those in which almost all entries are $\perp_{string}$. We can think of this type as, for example, *type person = [Name:string, spouse:person]*. Its elements certainly have finite representation, but are not compact elements of the domain of *person*. Therefore, we need to identify elements of the domains which have a finite representation. This identification must be done order-theoretically. Therefore, a proper definition of elements having a finite representation and identification of elements of solutions of recursive domain equations having finite representations remain open problems. We believe that progress towards solving these problems will suggests the right operations to be used for programming with recursive complex objects.

4.3 Bags and partial information

So far we have tacitly assumed that we deal with sets and duplicates are always removed. However, most practical database management systems use bags as the underlying data model. There has been some interest in languages for bags recently [GM93, LW94a, LW94b]. A standard bag language, called BQL or BALG, was obtained. It is supposed to play the same role for bags as (nested) relational algebra plays for set (or complex objects). One can also transfer the results on orderings from sets to bags. To define elementary updates, we should keep in mind that having a bag rather than a set means that each element of a bag represents an object and if there are many occurrences of some element, then at the moment certain objects are indistinguishable.

In view of this, we define updates on bags as follows. First, if b is an element of bag B, and $b \le b'$, then $B \overset{\mathrm{CWA}}{\leadsto}$ (and $\overset{\mathrm{OWA}}{\leadsto}$)$(B - \{\!|b|\!\}) \uplus \{\!|b'|\!\}$. Here $-$ is bag difference, $\uplus$ is additive union and $\{\!|\,|\!\}$ are bag brackets. In the case of OWA we also add $B \overset{\mathrm{OWA}}{\leadsto} B \uplus \{\!|b|\!\}$. Transitive closures of these relations are denoted by $\unlhd^{\mathrm{CWA}}$ and $\unlhd^{\mathrm{OWA}}$. It was shown by the author how to describe $\unlhd^{\mathrm{CWA}}$ and $\unlhd^{\mathrm{OWA}}$ algorithmically. For a finite bag B and an injective map $\phi : B \to \mathbb{N}$, also called *labeling*, by $\phi(B)$ we denote the set $\{(b, \phi(b)) \mid b \in B\}$. In other words, ϕ assigns a unique label to each element of a bag. If B is a finite bag of elements of a poset, then the ordering on pairs (b, n) where $b \in B$ and $n \in \mathbb{N}$ is the following: $(b, n) \le (b', n')$ iff $b \le b'$ and $n = n'$.

Proposition 29. (see also [LW95]) *The binary relations $\unlhd^{\mathrm{CWA}}$ and $\unlhd^{\mathrm{OWA}}$ on bags are partial orders. Given two bags B_1 and B_2, $B_1 \unlhd^{\mathrm{CWA}} B_2$ ($B_1 \unlhd^{\mathrm{OWA}} B_2$) iff there exist labelings ϕ and ψ on B_1 and B_2 respectively such that $\phi(B_1) \sqsubseteq^{\natural} \psi(B_2)$ (respectively $\phi(B_1) \sqsubseteq^{\flat} \psi(B_2)$).* □

That is, the correspondence between OWA and the Hoare ordering and CWA and the Plotkin ordering continues to hold.

We saw that that $\sqsubseteq^{\sharp}$ and $\sqsubseteq^{\flat}$ are definable in our basic set language $\mathcal{NRL}$. However, for bags the situation is different. It was shown in [LW95] that neither $\unlhd^{\mathrm{CWA}}$ nor $\unlhd^{\mathrm{OWA}}$ is definable in the basic bag language BQL. Hence, any implementation of a bag language that supports incomplete information must provide orderings at all types, as these can not be lifted from base types if powerful primitives like fixpoints are not used.

4.4 Language implementation

The core language for sets and or-sets has been implemented as a library of modules in Standard ML, see [GL94]. It was useful in several application such as querying incomplete design databases, or querying independent databases to obtain approximate answers. We believe that in the future implementations several changes must be made. For example, an algebraic syntax of *or-$\mathcal{NRL}$*, which is reflected by the syntax of OR-SML, should be changed to a more user friendly syntax, such as comprehensions [BLS+94]. This poses a few problems, such as incorporating normalization of disjunctive objects into the comprehension syntax. It is also important that a user be able to add any collection of null values to any preexisting type and define orderings on them. Currently this is possible only with user-defined new types. Finally, it would be interesting to see if using partial information leads to any new optimizations.

*Acknowledgements.*This paper is based on the results from my handwritten notes from 1992-1994 and a few results from two conference papers

[LW96, LW94b]. Many of the results can also be found in my thesis [Lib94]. While working on those notes and papers, I had an opportunity to discuss the results with a number of people, and the feedback I received from them was very helpful. I would like to thank Peter Buneman, Carl Gunter, Elsa Gunter, Achim Jung, Paris Kanellakis, Val Tannen, Victor Vianu and Limsoon Wong. I am also grateful to anonymous referee for helpful comments.

References

[AB86] S. Abiteboul and N. Bidoit. Non-first normal form relations: An algebra allowing data restructuring. *Journal of Computer and System Sciences*, 33(3):361–371, 1986.

[AKG91] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78:159–187, 1991.

[AM86] P. Atzeni and N. Morfuni. Functional dependencies and constraints on null values in database relations. *Information and Control*, 70(1):1-31, July 1986.

[BBW92] V. Breazu-Tannen, P. Buneman, and L. Wong. Naturally embedded query languages. In J. Biskup and R. Hull, editors, *LNCS 646: Proceedings of 4th International Conference on Database Theory, Berlin, Germany, October, 1992*, pages 140–154. Springer-Verlag, October 1992.

[BDW91] P. Buneman, S. Davidson, and A. Watters. A semantics for complex objects and approximate answers. *Journal of Computer and System Sciences*, 43(1):170–218, August 1991.

[Bis81] J. Biskup. A formal approach to null values in database relations. In *Advances in Data Base Theory: Volume 1*. Plenum Press, New York, 1981.

[BLS+94] P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, March 1994.

[BJO91] P. Buneman, A. Jung and A. Ohori. Using powerdomains to generalize relational databases. *Theoretical Computer Science*, 91:23–55, 1991.

[BBN91] V. Breazu-Tannen, P. Buneman, and S. Naqvi. Structural recursion as a query language. In *Proceedings of 3rd International Workshop on Database Programming Languages, Naphlion, Greece*, pages 9–19. Morgan Kaufmann, August 1991.

[BTS91] V. Breazu-Tannen and R. Subrahmanyam. Logical and computational aspects of programming with Sets/Bags/Lists. In *LNCS 510: Proceedings of 18th International Colloquium on Automata, Languages, and Programming, Madrid, Spain, July 1991*, pages 60–75. Springer Verlag, 1991.

[BW90] M. Barr and C. Wells. *Category Theory for Computing Science*. Series in Computer Science. Prentice Hall International, New York, 1990.

[Car88] L. Cardelli. Types for data-oriented languages. In J. W. Schmidt, S. Ceri, and M. Missikoff, editors, *LNCS 303: Advances in Database Technology — International Conference on Extending Database Technology, Venice, Italy, March 1988*. Springer-Verlag, 1988.

[Cod75] E. F. Codd. Understanding relations. *Bulletin of ACM SIGMOD*, pages 23–28, 1975.

[Cod79] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, December 1979.

[Col90] L. S. Colby. A recursive algebra for nested relations. *Information Systems*, 15(5):567–582, 1990.

[Day92] A. Day, The lattice theory of functional dependencies and normal decompositions. *Intern. J. of Algebra and Computation*, 2:409–431, 1992.

[DLM92] J. Demetrovics, L. Libkin and I. Muchnik. Functional dependencies in relational databases : a lattice point of view. *Discrete Applied Mathematics*, 40:155–185, 1992.

[Fag82] R. Fagin. Horn clauses and database dependencies. *Journal of ACM*, 29:952–985, 1982.

[GL94] E. Gunter and L. Libkin, OR-SML: A functional database programming language for disjunctive information and its applications. In D. Karagiannis, ed., *LNCS 856: Proceedings of Conference on Database and Expert Systems Applications, Athens, 1994*, Springer Verlag, 1994, pages 841–850.

[GM93] S. Grumbach and T. Milo. Towards tractable algebras for bags. *Proceedings of the 12th Conference on Principles of Database Systems*, Washington DC, 1993, pages 49–58.

[Gra77] J. Grant. Null values in relational databases. *Information Processing Letters*, 6:156–157, 1977.

[Gra91] G. Grahne. *The Problem of Incomplete Information in Relational Databases.* Springer-Verlag, Berlin, 1991.

[Gun85] C. Gunter. Comparing categories of domains. In *"Mathematical Foundations of Programming Semantics"* (A. Melton ed), Springer Lecture Notes in Computer Science, vol. 239, Springer, Berlin, 1985, pages 101–121.

[Gun92] C. Gunter. *Semantics of Programming Languages: Structures and Techniques.* Foundations of Computing. MIT Press, 1992.

[GZ88] G. Gottlob and R. Zicari. Closed world databases opened through null values. In *Proceedings of Very Large Databases*, pages 50–61, Cambridge, Massachusetts, 1988.

[IL84] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31:761–791, October 1984.

[INV91a] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete objects — a data model for design and planning applications. In J. Clifford and R. King, editors, *Proceedings of ACM-SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991*, pages 288–297. ACM Press, 1991.

[INV91b] T. Imielinski, S. Naqvi, and K. Vadaparty. Querying design and planning databases. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *LNCS 566: Deductive and Object Oriented Databases*, pages 524–545, Berlin, 1991. Springer-Verlag.

[JLP92] A. Jung, L. Libkin, and H. Puhlmann. Decomposition of domains. In *LNCS 598: Proceedings of 1991 Conference on Mathematical Foundations of Programming Semantics*, pages 235–258, Berlin, 1992. Springer-Verlag.

[Kan90] P. Kanellakis. Elements of relational database theory. In *Handbook of Theoretical Computer Science, Volume B*, pages 1075–1156. North Holland, 1990.

[Lib91] L. Libkin. A relational algebra for complex objects based on partial information. In J. Demetrovics and B. Thalheim, editors, *LNCS 495: Proceedings of Symposium on Mathematical Fundamentals of Database Systems, Rostock, 1991*, pages 36–41. Springer-Verlag, 1991.

[Lib92] L. Libkin. An elementary proof that upper and lower powerdomain constructions commute. *Bulletin of the EATCS*, 48:175–177, 1992.

[Lib94] L. Libkin. *Aspects of Partial Information in Databases.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1994.

[Lib95a] L. Libkin. Approximation in databases. In G. Gottlob and M. Vardi, editors, *LNCS 893: Proceedings of International Conference on Database Theory, Prague, 1995*, pages 411–424. Springer-Verlag, 1995.

[Lib95b] L. Libkin. Normalizing incomplete databases. In *Proceedings of 14th ACM Symposium on Principles of Database Systems*, pages 219–230, San Jose, CA, May 1995.

[Lip79] W. Lipski. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, 4(3):262–296, September 1979.

[Lip81] W. Lipski. On databases with incomplete information. *Journal of ACM*, 28:41–70, 1981.

[LL86] N. Lerat and W. Lipski. Nonapplicable nulls. *Theoretical Computer Science*, 46:67–82, 1986.

[LL90] M. Levene and G. Loizou. The nested relation type model: An application of domain theory to databases. *The Computer Journal*, 33:19–30, 1990.

[LL91] M. Levene and G. Loizou. Correction to "Null values in nested relational databases" by M. A. Roth, H. F. Korth, and A. Silberschatz. *Acta Informatica*, 28:603–605, 1991.

[LL93] M. Levene and G. Loizou. A fully precise null extended nested relational algebra. *Fundamenta Informaticae*, 19:303–343, 1993.

[LW94a] L. Libkin and L. Wong. New techniques for studying set languages, bag languages, and aggregate functions. In *Proceedings of 13th ACM Symposium on Principles of Database Systems*, pages 155–166, Minneapolis, Minnesota, May 1994.

[LW94b] L. Libkin and L. Wong. Some properties of query languages for bags. In C. Beeri, A. Ohori, and D. Shasha, editors, *Proceedings of 4th International Workshop on Database Programming Languages, New York, August 1993*, pages 97–114. Springer-Verlag, January 1994.

[LW94c] L. Libkin and L. Wong. Conservativity of nested relational calculi with internal generic functions. *Information Processing Letters*, 49:273–280, 1994.

[LW95] L. Libkin and L. Wong. On representation and querying incomplete information in databases with bags. *Information Processing Letters*, 56:209–214, 1995.

[LW96] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. *Journal of Computer and System Sciences*, 52(1):125–142, 1996.

[Oho90] A. Ohori. Semantics of types for database objects. *Theoretical Computer Science*, 76(1):53–91, 1990.

[PDGV89] J. Paredaens, P. De Bra, M. Gyssens and D. Van Gucht, *"The Structure of the Relational Data Model"*, Springer, Berlin, 1989.

[Plo76] G. D. Plotkin. A powerdomain construction. *SIAM Journal of Computing*, 5, September 1976.

[PS93] A. Poulovassilis and C. Small. A domain theoretic approach to integrating functional and logical database languages. In *Proceedings of VLBD*, pages 416–428, 1993.

[Rei78] R. Reiter. On closed world databases. In H. Gallaire and J. Minker, editors, *Logic and Databases.* Plenum Press, 1978.

[RKS89] M. A. Roth, H. F. Korth, and A. Silberschatz. Null values in nested relational databases. *Acta Informatica*, 26(7):615–642, 1989.

[Smy78] M. B. Smyth. Power domains. *Journal of Computer and System Sciences*, 16(1):23–36, 1978.

[SP94] D. Suciu and J. Paredaens. Any algorithm in the complex object algebra needs exponential space to compute transitive closure. In *Proceedings of 13th ACM Symposium on Principles of Database Systems*, pages 201–209, Minneapolis, Minnesota, May 1994.

[SS86] H.-J. Schek and M. H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[Suc94] D. Suciu. Fixpoints and bounded fixpoints for complex objects. In C. Beeri, A. Ohori, and D. Shasha, editors, *Proceedings of 4th International Workshop on Database Programming Languages, New York, August 1993*, pages 263–281. Springer-Verlag, January 1994.

[Tha89] B. Thalheim. On semantic issues connected with keys in relational databases permitting null values. *J. Inf. Process. and Cybernet.*, 25(1/2):11–20, 1989.

[Tha91] B. Thalheim. *"Dependencies in Relational Databases"*, Teubner-Texte zur Mathematik, Band 126, Stuttgart-Leipzig, 1991.

[TF86] S. J. Thomas and P. C. Fischer. Nested relational structures. In P. C. Kanellakis and F. P. Preparata, editors, *Advances in Computing Research: The Theory of Databases*, pages 269–307, London, England, 1986. JAI Press.

[Var86] M. Y. Vardi. On the integrity of databases with incomplete information. In *Proceedings of 5th ACM Symposium on Principles of Database Systems*, pages 252–266, 1986.

[Vas79] Y. Vassiliou. Null values in database management – a denotational semantics approach. In *Proceedings of SIGMOD*, 1979.

[Zan84] C. Zaniolo. Database relation with null values. *Journal of Computer and System Sciences*, 28(1):142–166, 1984.