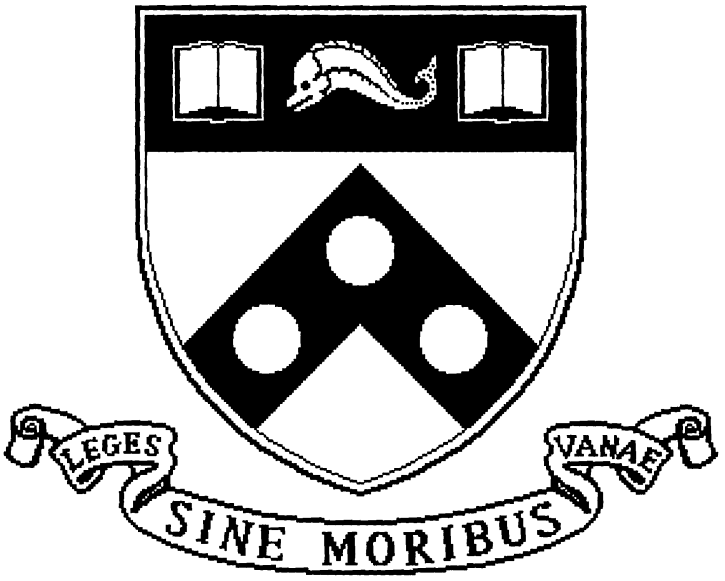
Approximation in Databases

MS-CIS-94-21
LOGIC & COMPUTATION 79

Leonid Libkin



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

May 1994

Approximation in Databases

Leonid Libkin*

Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA 19104-6389, USA
email: libkin@saul.cis.upenn.edu

Abstract

One source of partial information in databases is the need to combine information from several databases. Even if each database is complete for some "world", the combined databases will not be, and answers to queries against such combined databases can only be approximated. In this paper we describe various situations in which a precise answer cannot be obtained for a query asked against multiple databases. Based on an analysis of these situations, we propose a classification of constructs that can be used to model approximations.

One of the main goals is to show that most of these models of approximations possess universality properties. The main motivation for doing this is applying the data-oriented approach, which turns universality properties into syntax, to obtain languages for approximations. We show that the languages arising from the universality properties have a number of limitations. In an attempt to overcome those limitations, we explain how all the languages can be embedded into a language for conjunctive and disjunctive sets from [21], and demonstrate its usefulness in querying independent databases.

1 Introduction

The idea of using approximate answers to queries against databases with partial information has been known in the database literature for more than ten years. In his classical papers, Lipski [24, 25] suggested to use two approximations to answer queries Q for which a precise answer can not be found. The *lower approximation* to the answer to Q consists of those objects for we which one can conclude with certainty that they belong to the answer to Q. The *upper approximation* to the answer to Q consists of those objects for we which one can conclude that they may belong to the answer to Q.

However, it was not until ten years later that it was observed by Buneman, Davidson and Watters [5] that those pairs of approximations may not only be regarded as results of query evaluation but may also be used as a representation mechanism for certain kinds partial data. Moreover, this kind of partiality is different from traditional model such as null values and disjunctive information. If a query is asked against several databases, the combined databases may not be complete even if each database is complete for some "world". Hence, incompleteness shows up in the form of an *answer to query*, rather than the representation of data as in the classical models. Let us give some examples.

*Supported in part by NSF Grant IRI-90-04137 and AT&T Doctoral Fellowship.

Example: Querying independent databases

Simple approximations. The general problem of querying independent databases is the following: given a set of databases $D_1, \ldots, D_n$ and a query q that can not be answered by using information from one of D_i's, approximate the answer to q by using information from all $D_1, \ldots, D_n$. If it is impossible to answer q precisely, then the databases are divided into two groups, one giving the upper approximation to the answer to q and the other giving the lower approximations.

Consider the following problem. Suppose the university database has two relations, Employees and CS1 (for teaching the course CS1):

Employees

Name	Salary	Room
John	15K	$\perp$
Ann	17K	$\perp$
Mary	12K	$\perp$
Michael	14K	$\perp$

CS1

Name	Salary	Room
John	$\perp$	076
Michael	$\perp$	320

Assume that our query asks to compute the set TA of teaching assistants. We further assume that only TAs can teach CS1 and that every TA is a university employee. Also, for simplicity, we make an assumption that the Name field is a key. Of course this may not be the case, and solutions we consider work if no assumptions about keys were made. This assumption, however, makes the examples easier to understand. We also use nulls $\perp$ to make both relations have the same set of attributes.

Let us briefly outline how the TA query can be answered. We know that every person in CS1 is a TA; therefore, CS1 gives us the certain part of the answer. Moreover, every TA is an employee, hence finding people in the Employees relation who are not represented in the CS1 relation gives us the possible part of the answer to the TA query. Notice that it is possible to find possible TAs because Name is a key. If it were not, we would have to use *or-sets.*

A pair of relations CS1 and Employees is called a *sandwich* (for TA) [5]. The Employees relation is *an upper bound:* every TA is an Employee. The CS1 relation is *a lower bound:* every entry in CS1 represents a TA. We are looking for the set of TA – something that's in between; hence the name. Notice that in our example records in CS1 and Employees are *consistent:* for every record CS1, there is a record in Employees consistent with it. That is, they are joinable and their join can be defined. For example,

John	15K	$\perp$	$\vee$	John	$\perp$	076	$=$	John	15K	076

Hence, a sandwich (for a query Q) is a pair of relations R_1 and R_2 such that R_1 is an upper bound or an upper approximation to Q, R_2 is a lower bound or a lower approximation to Q, and R_1 and R_2 are consistent.

Assume a pair of consistent relations R_1 and R_2 is given. What is the semantics of the sandwich (R_1, R_2)? That is, what is the family of possible answers to Q which R_1 and R_2 approximate? To emphasize that R_1 is an upper approximation, we denote it by U from now on. Similarly, we denote the lower approximation R_2 by L.

To answer the question about semantics of (U, L) – at this stage, only informally – we appeal to the idea of representing partial objects as elements of ordered sets. In a graphical representation,

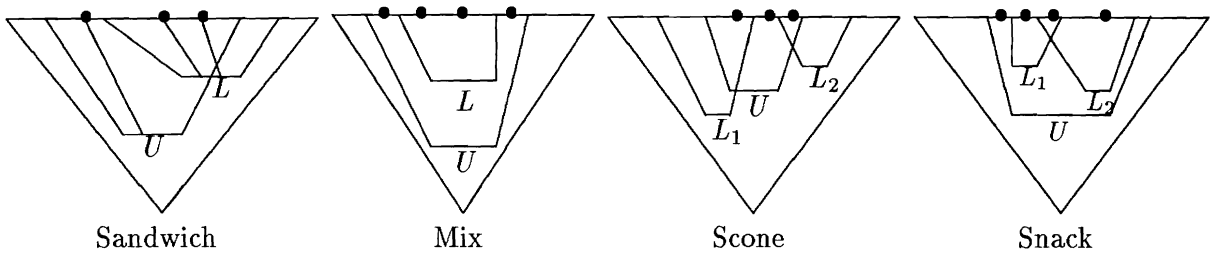


Figure 1: Models of approximations and their semantics

ordered sets will be shown as triangles standing on one of their vertices. That vertex represents the minimal, or bottom element. The side opposite to that vertex represents maximal elements. In our interpretation the order means "being less partial", or "being more informative". Then maximal elements correspond to complete descriptions, i.e. those that do not have any partial information at all.

The graphical representation of a sandwich (U, L) is shown in the first picture in figure 1. Trapezoids standing on U and L represent graphically elements of the whole space which are bigger than an element of U or L respectively. The semantics of a sandwich is a family of sets such as the one denoted by three bullets in the picture. There are two properties of such sets X that include them into the semantic space of a sandwich. First, for every element $l \in L$, there is an element $x \in X$ such that $l \leq x$. That is, each element of the lower approximation contributes to at least one element of X. Second, all X lies in the trapezoid standing on U, i.e. for every $x \in X$, there exists $u \in U$ such that $u \leq x$. That is, each element of X is represented by an element of the upper approximation.

Observe that in our particular example depicted in the picture, L is assumed to have two elements. Since both of them are under elements of the three-bullet set, which in turn are all above some elements of U, (U, L) satisfies the consistency condition, i.e. it is a sandwich.

Now, assume that the Name field is a key. Then we can replace certain nulls in relations CS1 and Employees by corresponding values, taken from the other relation. The reason is that certain tuples are joinable, and corresponding joins can be taken to infer missing values. One such join was shown above. Since Name is a key, we know that there is only one John and we assume that the same John is represented by both databases. Hence we infer that he is in the office 076 and his salary is 15K. Similarly for Michael we infer that he is in the office 320 and his salary is 14K.

We can regard the newly constructed relations as another approximation for TA. But this one satisfies a much stronger consistency condition than sandwiches: every record in the lower approximation is also found in the upper. We call a pair satisfying this consistency condition a *mix*. An example of a mix is shown in figure 1.

Mixes were introduced by Gunter [9] as an alternative approximation construct, whose properties are generally easier to study than properties of sandwiches because of its consistency condition in which no joins are involved. We shall discuss this phenomenon in details later.

Semantics of mixes is defined in exactly the same way as semantics of sandwiches: we look at sets that represent all elements of the lower approximation and whose elements are representable by the upper approximation. In Figure 1, a set shown by four bullets is such.

Approximating by many relations. Let us consider a more complicated situation. Assume now that CS1 has two sections: $CS1_1$ and $CS1_2$, and each section requires a teaching assistant. Assume that we have a pool of prospective TAs for each section that includes those graduate students who volunteered to be TAs for that section. Now suppose that the selection of TAs has been made, and those who have been selected were entered in the database of employees, while the database of prospective TAs remained unchanged. This situation can be represented by an example below:

Employees

Name	Salary	Room
John	15K	$\perp$
Ann	17K	$\perp$
Mary	12K	$\perp$
Michael	14K	$\perp$

$CS1_1$

Name	Salary	Room
John	$\perp$	076
Jim	$\perp$	$\perp$

$CS1_2$

Name	Salary	Room
Michael	$\perp$	320
Helen	$\perp$	451

Since all the selections have been made, at least one of prospective TAs for each section is now a TA, and therefore there is a record in Employees for him or her. That is, in each of the subrelations of CS1, at least one entry is consistent with the Employees relation.

Let us summarize the main difference between this construction and sandwiches or mixes.

1. The lower approximation is no longer a single relation but a *family of relations.*
2. The consistency condition does not postulate that all elements in the lower approximation are consistent with the upper approximation, but rather that there *exists* an element in each of the subrelations of the lower approximation that is consistent with the upper.

Such approximations are called *scones* [27]. We shall denote the lower approximation by $\mathcal{L}$ and its components by L_1, L_2 etc. The graphical representation of a scone with the two-element $\mathcal{L}$ is shown in Figure 1.

The semantics of a scone is a family of sets X that satisfy the following two properties. First, for every set $L \in \mathcal{L}$, there exist $l \in L$ and $x \in X$ such that $l \leq x$. Second, all X lies in the trapezoid standing on U. That is, for every $x \in X$, there exists $u \in U$ such that $u \leq x$. For example, in Figure 1 the set denoted by three bullets is such. Observe that the second property is exactly the same for scones as it is for sandwiches and mixes, but the first one is different and it reflects the difference in the structure of scones and sandwiches.

Now let us look at the data represented by $CS1_1$ and $CS1_2$. Assume again that the Name field is a key. Observe that some preprocessing can be done before any queries are asked. In particular, there is no entry for Jim in the Employees relation. Hence, Jim could not have been chosen as a possible TA for a section of CS1. Similarly, Helen can be removed from $CS1_2$. Having removed Jim and Helen from $CS1_1$ and $CS1_2$, we can now infer some of the null fields as we did before in order to obtain mixes from sandwiches. In the new approximation that we obtain, the condition expressing consistency of this approximation is much stronger than the condition we used for scones. In fact, all elements in $CS1_1$ and $CS1_2$ have become elements of Employees. In other words, taking into account that some entries can be nulls, we see that the new consistency condition says that every element of every set in the lower approximation is bigger than some element of the upper approximation. Such constructions are called *snacks*, see [26, 27]. The graphical representation of a snack with two-element $\mathcal{L}$ is given in Figure 1.

The semantics of snacks is defined precisely in the same way as the semantics of scones. For example, in Figure 1 the four-element set denoted by the bullets is in the semantics of $(U, \{L_1, L_2\})$. Thus, it is only the consistency condition that makes scones different from snacks.

Finally, what if we have arbitrary data coming from two independent databases that may not be consistent? For instance, there may be anomalies in the data that ruin various consistency conditions. Then we need a model that would not require any consistency condition at all. Such a model was introduced in [19]. Since it is in essence "all others put together", it is called *salad.*

The main problem that we address in this paper is *building the general theory of approximate answers to queries.* In particular, we address the following questions.

- What are the formal models of approximations? Is it possible to classify those models according to some general principle?
- Do approximation constructs corresponds to (a combination of) known datatypes?
- How do we program with approximations?

The paper is organized in follows. In section 2 we present preliminary results necessary to describe our approach. First we explain the approach to databases with partial information that treats database objects as subsets of some partially ordered space of descriptions. The meaning of the ordering is "being more informative". This approach is based on [6, 15, 18]. One of its important features is that it allows one to abstract from the concrete data model (e.g. relational, complex object) as it can be used with a variety of models, see [6, 18].

Then we explain the *data-oriented* paradigm for the query language design [7]. This approach is based on incorporating the operations *naturally* associated with datatypes into a query language. To find such operations, it is necessary to describe the semantic domains of those datatype via *universality properties.*

In section 3 we use the ordered semantics to give formal models of approximations and suggest a classification of those. The main part of the paper is section 4 in which we show that most of the constructs possess the universality properties. This tells us what the important operations on approximations are.

In section 5 we discuss programming with approximation. First, we apply the data-oriented paradigm to descriptions of approximations obtained in section 4. We discuss problems with using this approach, such as undecidability of certain preconditions that need to be checked to ensure well-definedness of programs. As a solution to this problem, we suggest using encoding approximation constructs with *or-sets* [13, 14, 30, 21] and explain how the language for or-sets from [21] is suitable for programming with approximations. In fact, a system based on this language [11] has been used in the problems of querying independent databases.

2 Preliminaries

2.1 Partial objects and ordered sets

Most models of partiality of data can be represented via orderings on values, e.g. [2, 32, 12, 8]. In [6, 18, 21] a general approach to the treatment of partial information in the context of ordered sets has been developed. Here we present the basics of that approach.

First, elements of base types are ordered. For example, if there is only one null value $\perp$, then the ordering is given by letting $\perp$ be less than any nonpartial value v. In the approach when we have three kinds of nulls — no information **ni**, existing unknown **un** and nonexisting **ne** — the ordering is given by $\mathbf{ni} < \mathbf{un} < v$ and $\mathbf{ni} < \mathbf{ne}$. For more examples, see [2, 6, 20].

Complex objects, or nested relations, are constructed from the base objects by using the record and the set type constructors. Therefore, one has to lift an order to records and sets. Lifting order to records does not pose a problem: it is done componentwise. For example, [Name: Joe, Age:$\perp$] $\leq$ [Name: Joe, Age: 28]. It is not immediately clear how to lift an order to sets. This problem also arises in the semantics of concurrency, where a number of solutions have been proposed, see [10]. Here we consider two, which turn out to be suitable for our problems. Given an ordered set $\langle A, \leq \rangle$, its subsets can be ordered by the *Hoare ordering* $\sqsubseteq^\flat$ or the *Smyth ordering* $\sqsubseteq^\sharp$:

$$X \sqsubseteq^\flat Y \Leftrightarrow \forall x \in X.\, \exists y \in Y.\, x \leq y$$

$$X \sqsubseteq^\sharp Y \Leftrightarrow \forall y \in Y.\, \exists x \in X.\, x \leq y$$

In early works on representing partiality via orders, the problem of choosing the right ordering has not been considered. Recently, in [21, 20] a theory for deciding which order is suitable for which collection was developed. It turns out that $\sqsubseteq^\flat$ is suitable for sets[1] and $\sqsubseteq^\sharp$ is suitable for or-sets [13, 14]. Or-sets, denoted by the angle brackets, are sets of exclusive possibilities, i.e. [Name: Joe, Age:$\langle 25, 27 \rangle$] says that Joe is 25 *or* 27 years old.

Orderings suggest a natural approach to the *semantics of partiality*: an object may denote any other object that is above it. For example, [Name: Joe, Age:$\perp$] denotes the set $\{$[Name: Joe, Age:n] $\mid$ $n \in \mathbb{N}\}$. Hence, we define the semantic function for the database objects of the same domain D as $[\![o]\!] = \{o' \in D \mid o' \geq o\}$. This semantics leads us to an important observation. Since sets are ordered by $\sqsubseteq^\flat$, then for any set X we have $[\![X]\!] = [\![\max X]\!]$, where $\max X$ is the set of maximal elements of X. Since or-sets are ordered by $\sqsubseteq^\sharp$, then for any or-set X we have $[\![X]\!] = [\![\min X]\!]$, where $\min X$ is the set of minimal elements of X. Elements of $\max X$ and $\min X$ are not comparable; such subsets of ordered sets are called *antichains*. Therefore, the ordered semantics suggests that the database objects are represented as antichains in certain posets, cf. [6, 18].

The idea of using orderings to represent partiality has been quite fruitful. A general theory of partial information and languages to handle it based on orderings was developed in [20]. As another example of its applicability, it was shown in [17] that a mistake in [29] discovered in [16] goes away if equivalence with respect to $\sqsubseteq^\flat$ is used instead of equality to define representation systems.

2.2 Data-oriented programming

In this subsection we give an overview of the *data-orientation* as a programming language paradigm (cf. Cardelli [7]) and demonstrate one instance of this approach: a language for sets.

It was observed in [7] that while traditional programming languages are mostly algorithmic and procedure-oriented and pay little attention to handling of data, dealing with information systems in general and databases in particular requires more emphasis on the data. Databases are designed

[1] Technically speaking, only if we believe in the open world assumption. For closed worlds, the Plotkin ordering [10] should be used. However, the nature of lower approximations, for which the set ordering will be used, suggests the open world assumption, so we consider only the Hoare ordering in this paper.

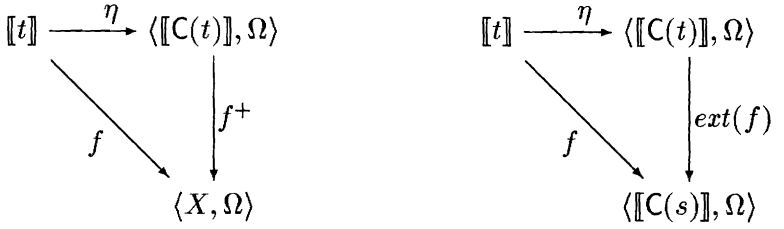


Figure 2: Structural recursion and *ext*

using some data models, e.g. relational, complex object, etc. To make it possible to program with data, it is necessary to represent the concept of a data model in a programming language. The best way to do it is to use *type systems* as a representation of data models.

Representing data models via type systems often allows *static type-checking* of programs which is particularly important in handling large data as run-time errors are very costly. To make sure that the type system is not too restrictive and does not limit the programmer's freedom, some form of polymorphism must be allowed. We allow all type constructs to be polymorphic, *e.g.* a set type constructor can be applied to any type, a product type constructor can be applied to any pair of types etc. For example, for a language for complex object, types are given by $t ::= b \mid [l_1 : t, \ldots, l_n : t] \mid \{t\}$.

It was suggested in [7] that one use *introduction* and *elimination* operations associated with a type constructor as primitives of a programming language. The introduction operations are needed to construct objects of a given type whereas the elimination operations are used to deconstruct them, or rather to do some computation with them. For example, for records, the introduction operation is forming a record with given fields, and the elimination operations are projections.

Since databases work with various kinds of collections, it is important to look at the introduction and elimination operations associated with those collections. One way to do it is to find operations that are *naturally* associated with collections. To do so, we define semantics of a collection type and try to characterize it by finding out if it has a *universality property.*

Universality properties immediately tell us what are the introduction and the elimination operations. Assume we have a collection type constructor that we denote by $\mathsf{C}(\cdot)$ and a type t. By *universality property* we mean that it is possible to find a set Ω of operations on the semantic domain of $\mathsf{C}(t)$, which we denote by $[\![\mathsf{C}(t)]\!]$, 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 2 commutes.

Hence, the introduction operations are η and those in Ω as we can use them to construct any object of type $\mathsf{C}(t)$ from objects of type t. The elimination operation is given by the universality property. In fact, the general elimination operation is the one that takes f into f^+.

At this point, let us see what these operations are for sets. The semantic domain of $\{t\}$ is the finite powerset of elements of t, that is, $\mathbb{P}_{\text{fin}}([\![t]\!])$. For any set X, its finite powerset $\mathbb{P}_{\text{fin}}(X)$ is the free semilattice generated by X. That is, $\eta(x) = \{x\}$, and operations of Ω are $\emptyset$ and $\cup$. The

operation that takes f into f^+ is the following (often called *structural recursion* [3]):

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

However, if e and u do not supply the range of f^+ with the structure of a semilattice, then f^+ may not be well-defined (see what happens if e is 0, f is $\lambda x.1$, and u is $+$.) The way to overcome this problem is to make sure that f^+ always lands in a domain supplied with the structure of a semilattice. The simplest way to ensure this 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 2.

The unique completing homomorphism is called $ext(f)$, the extension of f. Its semantics in the case of sets is $ext(f)\{x_1, \ldots, x_n\} = f(x_1) \cup \ldots \cup f(x_n)$ (that is, it "extends" f to sets.) This function is always well-defined. Using ext together with η, $\emptyset$, $\cup$, projections and record formation and the equality test gives us precisely the nested relational algebra [3] but the presentation is nicer than the standard ones, such as in [31]. This approach to the language design has proved extremely fruitful and allowed to solve some open problems (e.g. [23]) and develop languages for other collections (e.g. [21, 22]). Hence, we shall try to apply it to the approximation constructs. To do it, we first need formal models of approximation, and then the universality properties for those models.

Remark. There is mysticism in the diagrams above – these are constructions well known to mathematicians. The first one says that $[\![\mathsf{C}(t)]\!]$ is the free Ω-algebra generated by $[\![t]\!]$, or, in the category speak, establishes an *adjunction* between the category of Ω-algebras and the category where the semantic objects live. The second diagram represents going from that adjunction to the *Kleisli category of its monad*, see [1].

3 Formal models of approximations

In this section we re-examine the approximation constructs by applying the idea of representing database objects with partial information as elements of certain ordered sets. Before we do it, we need the notion of *consistency* in posets: two elements $x, y \in A$ are consistent (written $x \uparrow y$) if there exists $z \in A$ such that $x, y \leq z$. In the case of records, consistency means joinable (as in [33].) We shall use $\uparrow X$ for $\{y \mid y \geq x, \text{ some } x \in X\}$.

Recall the definition of a *sandwich.* It is given by an upper approximation U and a lower approximation L which satisfy the following consistency condition: for every $u \in U$, there is $l \in L$ such that u and l are consistent. Therefore, representing objects in approximating sets as elements of some posets, we can give a formal definition of sandwiches as follows:

Definition 1 (see [5].) *Given a poset $\langle A, \leq \rangle$, a* sandwich *over A is a pair of finite antichains (U, L) satisfying the following consistency condition: $\forall l \in L\ \exists u \in U : u \uparrow l$. The set U is usually referred to as the upper approximation and L as the lower approximation.*

The consistency condition for mixes says that every element in the lower approximation is at least as informative as some element of the upper. Hence, we arrive at

Definition 2 (see [9].) *Given a poset* $\langle A, \leq \rangle$, *a* mix *over* A *is a pair of finite antichains* (U, L) *satisfying the following consistency condition:* $\forall l \in L\ \exists u \in U :\ u \leq l$.

Now recall the definition of scones. In a scone, the lower approximation is a family of sets (relations), and the consistency condition says that for each set in the lower approximation, at least one element is consistent with an element of the upper approximation. Hence

Definition 3 (see [27].) *Given a poset* $\langle A, \leq \rangle$, *a* scone *over* A *is a pair* $(U, \mathcal{L})$ *where* U *is a finite antichain, and* $\mathcal{L} = \{L_1, \ldots, L_k\}$ *is a family of finite nonempty antichains which is itself an antichain with respect to* $\sqsubseteq^\sharp$. *That is,* $L_i \not\sqsubseteq^\sharp L_j$ *if* $i \neq j$. *In addition, a scone is required to satisfy the consistency condition:* $.\forall L \in \mathcal{L}\ \exists l \in L\ \exists u \in u :\ u \uparrow l$.

The last construction that we have seen was a snack. Snacks are obtained from scones in the same way as mixes are obtained from sandwiches: by using the assumption about keys, additional information is inferred. Thus, the consistency condition is similar to that of mixes.

Definition 4 (see [26, 27].) *Given a poset* $\langle A, \leq \rangle$, *a* snack *over* A *is a pair* $(U, \mathcal{L})$ *where* U *is a finite antichain, and* $\mathcal{L} = \{L_1, \ldots, L_k\}$ *is a family of finite nonempty antichains which is itself an antichain with respect to* $\sqsubseteq^\sharp$. *A snack is required to satisfy the consistency condition:* $\forall L \in \mathcal{L}\ \forall l \in L\ \exists u \in u :\ u \leq l$.

Now let us look at these constructs again. There are three main parameters that may vary and give rise to new constructs.

1. The lower approximation is either a set or a set of sets.

2. The consistency condition is of form

$$\begin{array}{rllll} & \mathbf{Q} l \in L & \exists u \in U & C(u,l) & \text{for simple lower approximations and} \\ \forall L \in \mathcal{L} & \mathbf{Q} l \in L & \exists u \in U & C(u,l) & \text{for multi-set lower approximations,} \end{array}$$

   where $\mathbf{Q}$ is a quantifier (either $\forall$ or $\exists$) and $C(u,l)$ is a condition that relates u and l.

3. The condition $C(u,l)$ is either $u \leq l$ or $u \uparrow l$.

Therefore, we have eight constructions since each of the parameters that may vary – the structure of the lower approximation, the quantifier $\mathbf{Q}$ and the condition $C(u,l)$ – has two possible values. For constructs that have a single set lower approximation we use notation $\mathcal{P}$ and for the constructs with multi-set lower approximation we use $\boldsymbol{\mathcal{P}}$. The rest is indicated in the superscript which consists of one or two symbols. The first is always a quantifier and indicates whether $\forall$ or $\exists$ is used as $\mathbf{Q}$. The second is omitted if the condition is $u \leq l$, and it is $\wedge$ if the condition is $u \uparrow l$ (to indicate that there is an element above u and l). Moreover, we have seen a need for constructs with no consistency condition, in order to deal with inconsistencies in independent databases. For such constructs we shall use just one superscript $\emptyset$.

Summing up, we have ten possible constructs: $\mathcal{P}^\forall, \boldsymbol{\mathcal{P}}^\forall, \mathcal{P}^{\forall\wedge}, \boldsymbol{\mathcal{P}}^{\forall\wedge}, \mathcal{P}^\exists, \boldsymbol{\mathcal{P}}^\exists, \mathcal{P}^{\exists\wedge}, \boldsymbol{\mathcal{P}}^{\exists\wedge}, \mathcal{P}^\emptyset, \boldsymbol{\mathcal{P}}^\emptyset$. For example, $\mathcal{P}^\forall(A)$ is the family of mixes over A, $\mathcal{P}^{\forall\wedge}(A)$ is the family of sandwiches over A, $\boldsymbol{\mathcal{P}}^\forall(A)$ is the family of snacks over A and $\boldsymbol{\mathcal{P}}^{\exists\wedge}(A)$ is the family of scones over A. This is summarized in the table below.

	type of consistency condition (quantifier-condition)				
L-part	$\forall \quad u \leq l$	$\forall \quad u \barwedge l$	$\exists \quad u \leq l$	$\exists \quad u \barwedge l$	no condition
one set	$\mathcal{P}^{\forall}$ (mix)	$\mathcal{P}^{\forall\wedge}$ (sandwich)	$\mathcal{P}^{\exists}$	$\mathcal{P}^{\exists\wedge}$	$\mathcal{P}^{\emptyset}$
family of sets	$\boldsymbol{\mathcal{P}}^{\forall}$ (snack)	$\boldsymbol{\mathcal{P}}^{\forall\wedge}$	$\boldsymbol{\mathcal{P}}^{\exists}$	$\boldsymbol{\mathcal{P}}^{\exists\wedge}$ (scone)	$\boldsymbol{\mathcal{P}}^{\emptyset}$

Order and semantics

Define two orderings, called *the Buneman orderings*, see [6, 9]. For pairs (U, L) and (U', L'), let

$$(U, L) \sqsubseteq^{\mathbb{B}} (U', L') \quad \text{iff} \quad U \sqsubseteq^{\sharp} U' \text{ and } L \sqsubseteq^{\flat} L'$$

In other words, $\sqsubseteq^{\mathbb{B}} = \sqsubseteq^{\sharp} \times \sqsubseteq^{\flat}$. For pairs $(U, \mathcal{L})$ and $(U', \mathcal{L}')$, let

$$(U, \mathcal{L}) \sqsubseteq_f^{\mathbb{B}} (U', \mathcal{L}') \quad \text{iff} \quad U \sqsubseteq^{\sharp} U' \text{ and } \forall L \in \mathcal{L}\ \exists L' \in \mathcal{L}' :\ L \sqsubseteq^{\sharp} L'$$

In other words, $\sqsubseteq_f^{\mathbb{B}} = \sqsubseteq^{\sharp} \times (\sqsubseteq^{\sharp})^{\flat}$. The index f is ised in $\sqsubseteq_f^{\mathbb{B}}$ to indicate that the ordering deals with families of sets in the lower approximations, whereas $\sqsubseteq^{\mathbb{B}}$ deals with simple lower approximations.

Claim. *The approximations must be ordered by the Buneman orderings.* □

The reader is referred to [19, 20] for the rationale behind this claim. It is justified by proving the results similar to those proved in [21, 20] for sets under open and closed world assumptions and or-sets and in [22] for bags.

Thus, when we consider approximation constructs $\mathcal{P}^i(A)$ and $\boldsymbol{\mathcal{P}}^i(A)$, where $i \in \{\forall, \exists, \forall\wedge, \exists\wedge, \emptyset\}$, we assume that they are ordered by $\sqsubseteq^{\mathbb{B}}$ and $\sqsubseteq_f^{\mathbb{B}}$ respectively.

Because of the limitation on the length of this document, we do not discuss the semantic here and only offer the definition of the semantic functions. The reader is invited to apply them to the examples in the introduction chapter and see that they corresponds to sets of TAs that can be approximated by Employees and CS1.

For simple approximations, we define $[\![(U, L)]\!] = \{X \in \mathbb{P}_{\text{fin}}(A) \mid U \sqsubseteq^{\sharp} X \text{ and } L \sqsubseteq^{\flat} X\}$. For constructions with multi-element lower approximations (like snacks and scones) the semantic function is given by $[\![(U, \mathcal{L})]\!] = \{X \in \mathbb{P}_{\text{fin}}(A) \mid U \sqsubseteq^{\sharp} X \text{ and } \forall i : \uparrow L_i \cap X \neq \emptyset\}$.

4 Universality properties of approximations

The flavor of the results

Before we give the results about universality of $\mathcal{P}^i(A)$ and $\boldsymbol{\mathcal{P}}^i(A)$, let us give a quick overview. The desired result would be to obtain the first diagram in figure 3, where $\eta(x) = (\{x\}, \{x\})$ for $\mathcal{P}^i(A)$ and $\eta(x) = (\{x\}, \{\{x\}\})$ for $\boldsymbol{\mathcal{P}}^i(A)$. That is, every *monotone* map f can be extended to a *monotone homomorphism* f^+. Unfortunately, this is not always possible and here is the reason. Let $x \barwedge y$ in A. Then $S_{xy} = (\{x\}, \{y\})$ is a sandwich and $\mathcal{S}_{xy} = (\{x\}, \{\{y\}\})$ is a scone. Thus, if $\mathcal{P}^{\forall\wedge}(A)$ or $\boldsymbol{\mathcal{P}}^{\exists\wedge}(A)$ were free algebras generated by A, there would be a way to construct S_{xy} and $\mathcal{S}_{xy}$ from the singletons $\eta(x)$. But this way must use the information about consistency in A and therefore can not be "universal"!

Therefore, we shall settle for less. Namely, we make the generating poset convey the information about consistency in A. We define the *consistent closure* of A as

$$A \barwedge A = \{(a, b) \mid a \in A, b \in A, a \barwedge b\}$$

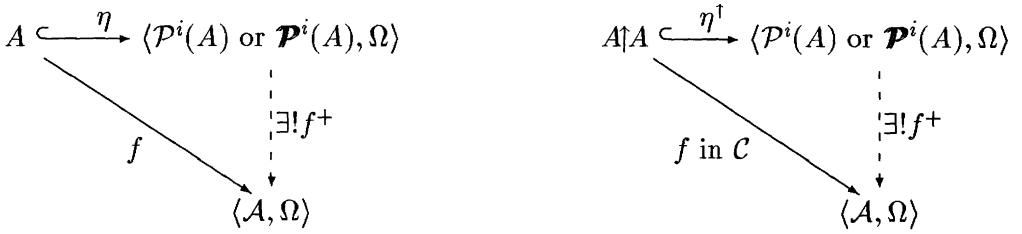


Figure 3: Universality results

The consistent closure of A can be embedded into $\mathcal{P}^i(A)$ and $\boldsymbol{\mathcal{P}}^i(A)$ (where $i \in \{\exists\wedge, \forall\wedge\}$) by means of the function $\eta^\uparrow(x,y) = (\{x\},\{y\})$ and $\eta^\uparrow(x) = (\{x\},\{\{y\}\})$. Since $A\restriction A$ interacts in a certain way with the structure of approximations, we shall seek the result like the one in the second diagram in figure 3. In this case we say that $\mathcal{P}^i(A)$ or $\boldsymbol{\mathcal{P}}^i(A)$ is freely-generated by $A\restriction A$ *with respect to the class $\mathcal{C}$ of monotone maps.*

We need the definition of three types of algebras, see [28]. Semilattices are algebras $\langle S, \cdot\rangle$ where $\cdot$ is a semilattice operation, i.e. idempotent, commutative and associative. A *bisemilattice* $\langle B, +, \cdot\rangle$ is an algebra with two semilattice operations. It is called *distributive* if both distributive laws holds. A *left normal band* $\langle B, *\rangle$ is an algebra with an idempotent associative operation $*$ such that $x*y*z = x*z*y$.

In what follows, we describe the algebras, then give the interpretation of their operations on the approximation constructs, and then present the results.

Universality of $\mathcal{P}^\forall(A)$ (mixes)

Algebra. A *mix* algebra $\langle M, +, \Box, e\rangle$ has partially ordered carrier M, one monotone binary operation $+$ and one monotone unary operation $\Box$. $\langle M, +, e\rangle$ is a semilattice with identity e, and in addition the following equations must hold: 1) $\Box(x+y) = \Box x + \Box y$, 2) $\Box\Box x = \Box x$, 3) $\Box x \leq x$, 4) $x + \Box x = x$, 5) $x + \Box y \leq x$.

Interpretation of operations. The ordering is interpreted as $\sqsubseteq^{\mathbf{B}}$. For the operations, $(U,L) + (V,M) = (\min(U \cup V), \max(L \cup M))$, $\Box(U,L) = (U,\emptyset)$ and $e = (\emptyset,\emptyset)$.

Theorem 1 ([9]) *$\mathcal{P}^\forall(A)$ is the free mix algebra generated by A.* □

Universality of $\mathcal{P}^{\forall\wedge}(A)$ (sandwiches)

Theorem 2 *For no Ω is $\mathcal{P}^{\forall\wedge}(A)$ the free ordered Ω-algebra generated by A.* □

However, we can overcome this by using the consistent closure and mix algebras with the same interpretation of operations.

Admissible functions. Let M be a mix algebra. A monotone map $f : A\restriction A \to M$ is called *admissible* (or sandwich-admissible) if $f(x,y) + f(z,y) \leq f(x,y)$ and $\Box f(x,y) = \Box f(x,z)$.

Theorem 3 *$\mathcal{P}^{\forall\wedge}(A)$ is the free mix algebra generated by $A \uparrow A$ with respect to the admissible maps.* □

Universality of $\mathcal{P}^{\exists}(A)$

Algebra. An algebra $\langle B, \oplus, * \rangle$ is called a *distributive bi-LNB algebra* if: 1) $\oplus$ and $*$ are left normal band operations. 2) All distributive laws between $*$ and $\oplus$ hold. 3) $a \oplus (b * c) = a \oplus b$. 4) $(a * b) \oplus b = (b * a) \oplus a$.

Order. $a \leq b := b \oplus a = a * b$.

Interpretation. $(U, L) \oplus (V, M) = (\min(U \cup V), L)$, $(U, L) * (V, M) = (U, \max(L \cup M))$.

Theorem 4 *$\mathcal{P}^{\exists}(A)$ is the free distributive bi-LNB algebra algebra generated by A.* □

Universality of $\mathcal{P}^{\emptyset}(A)$

Algebra. An algebra $\langle B, +, \Box, \Diamond \rangle$ is called a *bi-mix* algebra if $\langle B, +, \Box \rangle$ is a mix algebra, $x = \Box x + \Diamond x$ and $\langle B, +, \Diamond \rangle$ is a dual mix algebra. By this we mean that $\Diamond$ is a closure, that is, $\Diamond$ is monotone, $\Diamond x \geq x$, $\Diamond\Diamond x = \Diamond x$ and $\Diamond(x + y) = \Diamond x + \Diamond y$, and in addition $x + \Diamond x = x$ and $x + \Diamond y \geq x$.

Interpretation. Operations $+, \Box$ and e are interpreted as for mixes, and $\Diamond(U, L) = (\emptyset, L)$.

Theorem 5 *$\mathcal{P}^{\emptyset}(A)$ is the free bi-mix algebra generated by A.* □

Universality of $\boldsymbol{\mathcal{P}}^{\forall}(A)$ (snacks)

Algebra. A *snack* algebra is a bisemilattice $\langle B, +, \cdot \rangle$ in which $+$ has the identity e (i.e. $x + e = e + x = x$.) A snack algebra is ordered according to the $\cdot$ meet-semilattice operation.

Interpretation. $(U, \mathcal{L}) + (V, \mathcal{M}) = (\min(U \cup V), \max^\sharp(\mathcal{L} \cup \mathcal{M}))$ and $(U, \mathcal{L}) \wedge (V, \mathcal{M}) = (\min(U \cup V), \max^\sharp\{\min(L \cup M) \mid L \in \mathcal{L}, M \in \mathcal{M}\})$ where $\max^\sharp$ means family of maximal elements w.r.t. $\sqsubseteq^\sharp$. e is interpreted as $(\emptyset, \{\emptyset\})$.

Theorem 6 (see also [28, 27]) *$\boldsymbol{\mathcal{P}}^{\forall}(A)$ is the free snack algebra generated by A.* □

Universality of $\boldsymbol{\mathcal{P}}^{\forall\wedge}(A)$

Theorem 7 *Let Ω_+ be a set of operations on elements of $\boldsymbol{\mathcal{P}}^{\forall\wedge}(A)$ such that $+$ is a derived operation. Then $\boldsymbol{\mathcal{P}}^{\forall\wedge}(A)$ is* not *the free ordered Ω_+-algebra generated by A.* □

Universality of $\boldsymbol{\mathcal{P}}^{\exists}(A)$

Theorem 8 *Let Ω_+ be a set of operations on elements of $\boldsymbol{\mathcal{P}}^{\exists}(A)$ such that $+$ is a derived operation. Then $\boldsymbol{\mathcal{P}}^{\exists}(A)$ is* not *the free ordered Ω_+-algebra generated by A.* □

Universality of $\mathcal{P}^{\exists\wedge}(A)$ (scones)

Algebra. A *scone* algebra is an algebra $\langle Sc, +, *, e\rangle$ where $+$ is a semilattice operation with identity e, $*$ is a left normal band operation, $+$ and $*$ distribute over each other, the absorption laws hold and $e * x = e$. In other words, a scone algebra is an "almost distributive lattice" – commutativity of one of the operations is replaced by the law of the left normal bands.

Order. $x \cdot y = x * y + y * x$ is a semilattice operation and the order on scone algebras is defined by $x \leq y$ iff $x \cdot y = x$.

Interpretation. Operations $+$ and e are interpreted as for snacks and $(U, \mathcal{L}) * (V, \mathcal{M}) = (U, \max^\sharp\{\min(L \cup M) \mid L \in \mathcal{L}, M \in \mathcal{M}\})$.

Admissibility. Let $\langle Sc, +, *, e\rangle$ be a scone algebra. A monotone map $f : A{\restriction}A \to Sc$ is called *admissible* if $f(u,l) * f(v,m) = f(u,m) * f(w,l)$ and $f(u,l) * e = f(u,m) * e$.

A monotone function $f : A \to Sc$ from a poset A to a scone algebra Sc is called *scone-admissible* if, for any two consistent pairs $x{\restriction}y_1$ and $x{\restriction}y_2$ such that $x, y_i \leq z_i, i = 1, 2$, the following holds:

$$(f(x) * e + f(z_1)) * f(y_1) * f(y_2) = (f(x) * e + f(z_2)) * f(y_1) * f(y_2)$$

Theorem 9 *1) $\mathcal{P}^{\exists\wedge}(A)$ is the free scone algebra generated by $A{\restriction}A$ with respect to the admissible maps.*
2) $\mathcal{P}^{\exists\wedge}(A)$ is the free scone algebra generated by A with respect to the scone-admissible maps.
3) Let Ω_{Sc} be a set of operations on scones such that $+, *$ and e are derived operations. Then $\mathcal{P}^{\exists\wedge}(A)$ is not *the free ordered Ω_{Sc}-algebra generated by A.* □

Universality of $\mathcal{P}^{\emptyset}(A)$

Algebra. A *salad algebra* $\langle Sd, +, \cdot, \Box, \Diamond\rangle$ is an algebra with two semilattice operations $+$ and $\cdot$ and two unary operation $\Box$ and $\Diamond$ such that the following equations hold: 1) $x \cdot (y + z) = x \cdot y + x \cdot z$. 2) $x = \Box x + \Diamond x$. 3) $\Box(x + y) = \Box x + \Box y = \Box x \cdot \Box y = \Box(x \cdot y)$. 4) $\Diamond(x + y) = \Diamond x + \Diamond y$. 5) $\Diamond(x \cdot y) = \Diamond x \cdot \Diamond y$. 6) $\Box x \cdot \Diamond y = \Box x$. 7) $\Diamond x \cdot \Diamond y + \Diamond x = \Diamond x$. 8) $\Diamond\Diamond x = \Diamond x$. 9) $\Box\Box x = \Box x$.

Interpretation. Operations $+$ and $\cdot$ are interpreted as for snacks, and $\Box$ and $\Diamond$ as for $\mathcal{P}^{\emptyset}(A)$.

Theorem 10 *$\mathcal{P}^{\emptyset}(A)$ is the free salad algebra generated by A.* □

5 Programming with approximations

In this section we consider programming with approximations. First, we turn the universality properties from section 4 into programming syntax. We then show that this approach has a number of drawbacks. In an attempt to overcome those problems, we look at the semantic connection between approximations and sets and or-sets, that suggests an encoding of the approximation constructions. We use the encodings and the language *or-*$\mathcal{NRL}$ of [21] to show how a number of typical problems can be solved.

We also would like to remark that this approach is not purely theoretical. It was used in the system OR-SML [11] (which is the Standard ML enhanced with the types of complex objects and or-sets and some features of DBPLs) to query independent databases.

$$
\begin{array}{llcl}
fun & f^+(\emptyset,\emptyset) & = & e \\
| & f^+(\eta(x)) & = & f(x) \\
| & f^+(M_1 + M_2) & = & u(f^+(M_1), f^+(M_2)) \\
| & f^+(\Box M) & = & h(f^+(M))
\end{array}
\qquad
\begin{array}{llcl}
fun & f^+(\emptyset,\emptyset) & = & e \\
| & f^+(\eta^\uparrow(x,y)) & = & f(x,y) \\
| & f^+(S_1 + S_2) & = & u(f^+(S_1), f^+(S_2)) \\
| & f^+(\Box S) & = & h(f^+(S))
\end{array}
$$

Figure 4: Structural recursion on mixes and sandwiches

Using universality properties

Because of the space limitations, we consider only mixes and sandwiches. We consider them as *type constructors*. That is, for any type t we now have a new type $t\ mix$ such that $[\![t\ mix]\!] = \mathcal{P}^{\forall}([\![t]\!])$ and a new type $t\ sand$ such that $[\![t\ sand]\!] = \mathcal{P}^{\forall\wedge}([\![t]\!])$. Since mixes possess universality property, we can define the structural recursion on them. Similarly, the structural recursion can be defined on sandwiches, but the second clause must be different since sandwiches are generated by $A\uparrow A$ rather than A. See figure 4.

The structural recursion has a number of parameters: in addition to f, they include e, u and h prescribing its action in all possible cases of constructing a new mix/sandwich. Similarly to the case of sets, one might ask if, by setting these parameters in such a way that they do not obey the law of the equational theory, one may write ill-defined programs. This is indeed the case. In fact,

Theorem 11 *It is undecidable whether the structural recursion on mixes or sandwiches is well-defined for a given choice of e, u and h.* □

The solution that worked for sets was to impose syntactic restrictions on the general form of structural recursion. In the case of mixes a similar restriction yields the following construct: $mix\_ext(f) \stackrel{\text{def}}{=} f^+[(\emptyset,\emptyset), f, +, \Box]$ provided f sends elements of type t to $s\ mix$. In this case $mix\_ext(f)$ is a function of type $t\ mix \to s\ mix$. However, this alone does not eliminate the need to verify preconditions in the case when we use the ordered semantics. Functions agree with the ordered semantics iff they are monotone (see [20] and appendix.) Thus, monotonicity of f is needed for well-definedness of *mix_ext*. Now observe that if we disregard the second components in mixes, then we obtain the structural recursion on sets. Hence, its restriction to the *ext* operator gives us the nested relational algebra. However,

Theorem 12 *If sets are ordered by $\sqsubseteq^\flat$, then it is undecidable whether the semantics of an expression in the nested relational algebra is a monotone function.* □

We can observe the same phenomenon for other approximations. Thus, the data-oriented approach to programming with approximations has a number of problems. First, most operations used in the universality properties for approximations are not as intuitive as union, intersection and so on. Therefore, the average programmer would have a very hard time trying to write a program that uses constructs like f^+ in figure 4. Second, all approximations have different equational characterizations, and therefore there are several forms of structural recursion and as many sets of the monad primitives. This means that the language must contain all of them and therefore it is going to be too complicated to comprehend even for a theoretician, let alone a

programmer. Furthermore, in many applications more than one approximation model is used, and therefore in addition to ten approximations we also need a few dozen of operations that coerce one approximation into another. Finally, verification of preconditions remains a big problem, and it can not be taken care of by the compiler as the preconditions are undecidable – even for the monad operations when the ordered model is used. Therefore, we need a unifying framework for programming with approximations.

Using or-sets

Recall that or-sets are sets of disjunctive possibilities: an or-set $\langle 1,2,3\rangle$ denotes an integer which is 1, or 2 or 3. A language *or-*$\mathcal{NRL}$ was proposed in [21]. Its type system includes, in addition to sets and records, the or-set type constructor $\langle t\rangle$. Its expressions include those in the nested relational algebra and an or-set analog for each set operation. In addition, there is an operation $\alpha : \{\langle t\rangle\} \to \langle\{t\}\rangle$ which essentially converts a conjunctive normal form into disjunctive normal form.

The reason we suggest using or-sets to program with approximations is intuitively the following. If we look at how approximations are ordered, and then recall that $\sqsubseteq^\flat$ is used for sets and $\sqsubseteq^\sharp$ is used for or-sets, then this suggests the following encoding of the approximation constructs:

Approximations	Encoding
t mix, *t sand* and similar	$\langle t\rangle \times \{t\}$
t snack, *t scone* and similar	$\langle t\rangle \times \{\langle t\rangle\}$

In fact, there is a very close semantic connection between or-sets and approximations that further justifies this connection, see [20].

To show that this encoding can indeed be used to program with approximations, denote by $\mathcal{L}_\star$ the language obtained from the restricted form of structural recursion (that is, *ext*) for each approximation $\star$ for which we have established a universality property. Then we have

Theorem 13 *Using encoding of approximation constructs with sets and or-sets, the following can be expressed in* or-$\mathcal{NRL}$.

1. *All operations on approximations arising from the universality properties.*
2. *Orderings on approximations and tests for the consistency conditions.*
3. *All languages* $\mathcal{L}_\star$. □

Example: removal of anomalies and promotion in sandwiches

As an example of using the encoding with sets and or-sets, let us show how two of the algorithms from [5] can be implemented. Instead of using the abstract syntax of *or-*$\mathcal{NRL}$, we use the syntax of OR-SML, an SML based DBPL. In the following data `emp` and `cs1` we have two problems. First, the consistency condition does not hold (since Jim is not in `emp`.) Second, using the assumption about keys, values of some attributes can be inferred.

Here we use empty sets to represent null values (other solutions are possible). Note that `emp` is an or-set and `cs1` is a set. `co` is the OR-SML type of complex objects. `compat` checks if two records are compatible (together with `big_meet`, `join`, `ormember` and others it comes from the standard

library.) Then the first function (**anomaly**) removes the anomaly (Jim) and the second (**promote**) infers values of missing attributes.

```
- val emp = <('John', ({10}, {})), ('Mary', ({12}, {}))> : co
- val cs1 =   {('John', ({}, {76})), ('Jim', ({}, {320}))} : co

- fun anomaly compat (R,S) = let fun compat_to_X (X,x) =
   ormember(mkboolco(true),(orsmap (fn z => compat(z,x)) X))
in (R, select (fn z => compat_to_X (R,z)) S) end;
- fun promote compat (R,S) = let fun compat_to_x (X,x) =
      orselect (fn z => compat(z,x)) X
in (R,alpha(smap (fn z => big_meet
    (orflat(orsmap (fn v => join(z,v)) (compat_to_x (R,z))))) S)) end;

- val (R2,S2) = promote compatible (anomaly compatible (emp, cs1));

val R2 = <('John', ({10}, {})), ('Mary', ({12}, {}))> : co
val S2 = <{('John', ({10}, {76}))}> : co
```

Thus, this solution tells us that John from office 76 is a TA with salary 10K, and Mary with salary 12K could be a TA. Hence the result is an approximation in the sense of Lipski: we have the set of "for sure" answers and the set of "maybe" answers.

6 Conclusion

All existing papers on approximate answers to queries against independent databases ([5, 9, 26, 27]) did not address two important problems, which we have to look at in order to build a general theory. The first problem is a classification of models. In each of the above mentioned papers, only one or two models are considered, even though it is clear they do not cover all possible situations. The second problem is programming with the approximation constructs. In its rudimentary form this problem was considered in [5] who proposed the **promote** operation, but no general principles are known.

Our goal was to address these two problems. Let us briefly summarize what has been achieved.

- Using the approach to partial information based on represent partiality via order on objects (cf. [5, 18, 20]), we have given formal models of approximate answers to queries and classified those, coming up with ten possible constructs.
- We have explained a new approach to the query language design, based on turning universality properties into syntax, thus obtaining the introduction and elimination operations for the data types. Applying this approach to approximations tells us what the operations *naturally* associated with the constructs are. In order to do so, we have characterized most of the approximation constructs via their universality properties.
- We have looked at the languages arising from the universality properties of approximations, and showed that they have three major limitations: the operations are rather hard to grasp,

there are too many of them and the compiler can not verify the preconditions for well-definedness. To overcome these problems, we suggested using or-sets to encode approximations, and showed how the language from [11, 21] can be useful in answering a typical query against independent databases.

Acknowledgements. I wish to thank Peter Buneman, Carl and Elsa Gunter, Paris Kanellakis, Hermann Puhlmann, Anna Romanowska and especially Achim Jung for their help.

References

[1] M. Barr and C. Wells, *"Category Theory for Computing Science"*, Prentice Hall, 1990.

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

[3] V. Breazu-Tannen, P. Buneman, and L. Wong. Naturally embedded query languages. In *LNCS 646: Proc. ICDT, Berlin, Germany, October, 1992*, pages 140–154. Springer-Verlag, October 92.

[4] V. Breazu-Tannen and R. Subrahmanyam. Logical and computational aspects of programming with sets/bags/lists. In *LNCS 510: Proc. of 18th ICALP, Madrid, Spain, July 1991*, pages 60–75. Springer Verlag, 1991.

[5] P. Buneman, S. Davidson, A. Watters, A semantics for complex objects and approximate answers, *JCSS* 43(1991), 170–218.

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

[7] L. Cardelli. Types for data-oriented languages. In *Proceedings of EDBT-88* (J.W. Schmidt, S. Ceri and M. Missikoff eds), Springer Lecture Notes in Computer Science, vol. 303, Springer Verlag, 1988.

[8] G. Grahne, *"The Problem of Incomplete Information in Relational Databases"*, Springer, Berlin, 1991.

[9] C. Gunter, The mixed powerdomain, *Theoretical Computer Science* 103 (1992), 311–334.

[10] C. Gunter, *"Semantics of Programming Languages"*, The MIT Press, 1992.

[11] E. Gunter and L. Libkin, OR-SML: A functional database programming language for disjunctive information and its applications. *5th International Conference on Database and Expert Systems Applications*, 1994, to appear.

[12] T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM* 31(1984), 761–791.

[13] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete objects — a data model for design and planning applications. In *Proc. of ACM-SIGMOD, Denver, Colorado, May 1991*, pages 288–297.

[14] T. Imielinski, S. Naqvi, and K. Vadaparty. Querying design and planning databases. In *LNCS 566: Deductive and Object Oriented Databases*, pages 524–545, Berlin, 1991. Springer-Verlag.

[15] A. Jung, L. Libkin and H. Puhlmann, Decomposition of domains, In: *Proceedings of the Conference on Mathematical Foundations of Programming Semantics–91*, Springer LNCS 598, Springer Verlag, Berlin, 1992, pages 235–258.

[16] 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 (1991), 603-605.

[17] M. Levene and G. Loizou, A fully precise null extended nested relational algebra. *Fundamenta Informaticae* 19 (1993), 303-343.

[18] L. Libkin, A relational algebra for complex objects based on partial information, In *LNCS 495: Proceedings of MFDBS-91*, pages 36–41, Rostock, 1991. Springer-Verlag.

[19] L. Libkin, Algebraic characterization of edible powerdomains, Technical Report MS-CIS-93-70/L&C 71, University of Pennsylvania, 1993.

[20] L. Libkin. *"Aspects of Partial Information in Databases"*. PhD Thesis, University of Pennsylvania, 1994.

[21] L. Libkin and L. Wong, Semantic representations and query languages for or-sets, *Proceedings of the 12th Conference on Principles of Database Systems,* Washington, DC, May 1993, pages 37–48.

[22] L. Libkin and L. Wong, Some properties of query languages for bags, In *Proceedings of the Fourth Workshop on Database Programming Languages, Manhattan NY, August 30–September 1, 1993*, Springer Verlag, 1994, pages 97–114.

[23] L. Libkin and L. Wong, New techniques for studying set languages, bag languages and aggregate functions, In *PODS-94*, to appear.

[24] W. Lipski, On semantic issues connected with incomplete information in databases, *ACM Trans. Database Systems* 4 (1979), 262–296.

[25] W. Lipski, On databases with incomplete information, *J. ACM* 28 (1981), 41–70.

[26] T.-H. Ngair. *"Convex Spaces as an Order-theoretic Basis for Problem Solving"*, (PhD Thesis), Technical Report MS-CIS-92-60, University of Pennsylvania, 1992.

[27] H. Puhlmann, The snack powerdomain for database semantics, In *LNCS 711: MFCS-93,* (A. Borzyszkowski ed.), Springer Verlag, 1993, pages 650–659.

[28] A. Romanowska and J.D.H. Smith, *"Modal Theory: An Algebraic Approach to Order, Geometry and Convexity"*, Heldermann Verlag, Berlin, 1985.

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

[30] B. Rounds, Situation-theoretic aspects of databases, In *Proceedings of Conference on Situation Theory and Applications*, CSLI vol. 26, 1991, pages 229-256.

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

[32] Y. Vassiliou, Null values in database management – a denotational semantics approach. In: *SIGMOD 1979*, pages 162–169.

[33] C. Zaniolo. Database relations with null values. *JCSS* 28 (1984), 142–166.

APPENDIX

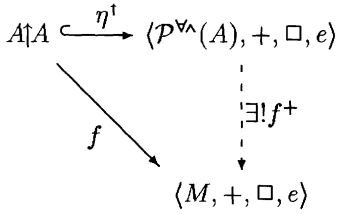
Proofs of theorems

Notation. In proofs we often omit the set brackets $\{\}$ when we deal with singletons. In particular, by $\{x\}$ we mean a family of sets that consists of one singleton. We shall also occasionally omit commas separating elements of sets, writing xyz for $\{x, y, z\}$.

Proof of theorem 2. Assume that there exists a set of operation Ω such that $\mathcal{P}^{\forall\wedge}(A)$ the free ordered Ω-algebra generated by A for any poset A. Let $A = \{x, y, z\}$ be an antichain and $A' = \{x', y', z'\}$ be a poset such that $x', y' \precsim z'$ and $x' \not\precsim y'$, $y' \not\precsim x'$. Let $f : A \to \mathcal{P}^{\forall\wedge}(A')$ be defined by $f(a) = (a', a'), a \in A$. Now the assumed universality property tells us that f can be extended to a monotone Ω-homomorphism $f^+ : \mathcal{P}^{\forall\wedge}(A) \to \mathcal{P}^{\forall\wedge}(A')$. Let $\mathcal{S} \in \mathcal{P}^{\forall\wedge}(A')$. Since $\mathcal{P}^{\forall\wedge}(A')$ is the free Ω-algebra generated by A', we can find a term t in the signature Ω such that $\mathcal{S} = t(\eta(x'), \eta(y'), \eta(z'))$. Since $\eta(x') = f(x) = f^+(\eta(x))$ and similarly for y' and z', we obtain $\mathcal{S} = f^+(t(\eta(x), \eta(y), \eta(z))) = f^+(\mathcal{S}_0)$ for some $\mathcal{S}_0 \in \mathcal{P}^{\forall\wedge}(A)$. Therefore, f^+ is onto.

Define $\mathcal{P}^{\forall\wedge}_{\neg xy}(A)$ as the set of elements of $\mathcal{P}^{\forall\wedge}(A)$ which are not under (x, x) or (y, y). It is easy to check that $\mathcal{P}^{\forall\wedge}_{\neg xy}(A)$ includes the following: (z, z), (xz, z), (yz, z), $(z, \emptyset)$, (xz, xz), (yz, yz), (xy, xy), (xyz, xz), (xyz, yz), (xyz, xy), (xyz, z). Similarly, define $\mathcal{P}^{\forall\wedge}_{\neg x'y'}(A')$ as the set of elements of $\mathcal{P}^{\forall\wedge}(A')$ which are not under (x', x') or (y', y'). These are: (x', y'), (y', x'), $(x'y', z')$, $(z', x'y')$, (x', z'), (z', x'), (y', z'), (z', y'), $(z', \emptyset)$, (z', z'). Since f^+ is monotone, we derive that its restriction on $\mathcal{P}^{\forall\wedge}_{\neg xy}(A)$ must be an onto map from a subset of $\mathcal{P}^{\forall\wedge}_{\neg xy}(A)$ to $\mathcal{P}^{\forall\wedge}_{\neg x'y'}(A')$. Observe that in $\mathcal{P}^{\forall\wedge}_{\neg xy}(A)$ the only element that is not above (xyz, z) is $(z, \emptyset)$. Hence, if $f^+((xyz, z)) = \mathcal{S} \in \mathcal{P}^{\forall\wedge}_{\neg x'y'}(A')$, then $f^+(\mathcal{P}^{\forall\wedge}_{\neg xy}(A) - \{(z, \emptyset)\})$ is a subset of the principal filter of $\mathcal{S}$ in $\mathcal{P}^{\forall\wedge}_{\neg x'y'}(A')$. However, $\mathcal{P}^{\forall\wedge}_{\neg x'y'}(A')$ has four minimal elements: (x', y'), (y', x'), $(x'y', z')$ and $(z', \emptyset)$ which shows that f^+ can not be an onto monotone map between $\mathcal{P}^{\forall\wedge}_{\neg xy}(A)$ and $\mathcal{P}^{\forall\wedge}_{\neg x'y'}(A')$. This contradiction shows that $\mathcal{P}^{\forall\wedge}(A)$ can not be obtained as the free Ω-algebra generated by A. □

Proof of theorem 3. We must show that, given a mix algebra M and an admissible map $f : A\mathrel{|}A \to M$, there exists a unique mix homomorphism $f^+ : \mathcal{P}^{\forall\wedge}(A) \to M$ such that the following diagram commutes:



Proof. We omit an easy verification that $\mathcal{P}^{\forall\wedge}(A)$ is a mix algebra.

Let us first establish a number of useful properties of admissible maps. In what follows, f is always an admissible map from $A\mathrel{|}A$ to M.

1) Assume $v \precsim u$ and $u\mathrel{|}l$. Then $f(u, l) + f(v, l) = f(v, l)$.

First, $f(u, l) \geq f(v, l)$. By monotonicity of $+$, $f(v, l) = f(v, l) + f(v, l) \leq f(v, l) + f(u, l)$. But since f is admissible, $f(u, l) + f(v, l) \leq f(v, l)$. Hence, 1) holds.

2) Assume $p \succsim l$, $v \uparrow l$ and $q \uparrow p$. Then $f(v,l) + f(q,p) = \Box f(v,v) + f(q,p)$.

First show $f(q,p) + f(q,l) = f(q,p)$. By monotonicity, $f(q,p) + f(q,l) \leq f(q,p) + f(q,p) = f(q,p)$. On the other hand, $f(q,p) + f(q,l) \geq f(q,p) + \Box f(q,l) = f(q,p) + \Box f(q,p) = f(q,p)$, which proves the equation. Since $\Box f(v,v) = \Box f(v,l) \leq f(v,l)$, the $\geq$ inequation for 2) holds. Conversely, $f(v,l) + f(q,p) = f(v,l) + f(q,l) + f(q,p) = \Box f(v,l) + f(v,l) + f(q,l) + f(q,p) \leq \Box f(v,l) + f(q,l) + f(q,p) \leq \Box f(v,v) + f(q,p)$ which shows the reverse inequation. 2) is proved.

3) If $l \precsim m$, then $f(v,l) + f(q,m) = \Box f(v,v) + f(q,m)$.

The $\geq$ inequation is obvious. As in the proof of 2), we obtain $f(v,l) + f(q,m) = f(v,l) + f(q,l) + f(q,m) = \Box f(v,l) + f(v,l) + f(q,l) + f(q,m) \leq \Box f(v,l) + f(q,l) + f(q,m) \leq \Box f(v,l) + f(q,m) = \Box f(v,v) + f(q,m)$.

4) Assume $v \precsim u$. Then $f(v,l) = f(u,l) + \Box f(v,v)$.

First, $f(u,l) + f(v,l) \leq f(v,l) = f(v,l) + f(v,l) \leq f(u,l) + f(v,l)$; hence $f(u,l) + f(v,l) = f(v,l)$. Now we have: $f(v,l) = f(v,l) + f(u,l) \geq f(u,l) + \Box f(v,l) = f(u,l) + \Box f(v,v)$. On the other hand, $f(v,l) = f(v,l) + \Box f(v,l) \leq f(u,l) + \Box f(v,v)$, proving 4).

5) If $v \succsim u$, then $\Box f(u,u) + \Box f(v,v) = \Box f(v,v)$.

According to the proof of 4), $f(u,v) + f(v,v) = f(v,v)$ and from this 5) follows immediately.

6) Assume $u \uparrow l$ and $v \uparrow l$. Then $f(v,l) + \Box f(u,u) = f(v,l) + \Box f(u,u) + f(u,l)$.

Since $\Box f(u,u) = \Box f(u,l) \leq f(u,l)$, the $\leq$ inequality holds. Since $f(v,l) + f(u,l) \leq f(v,l)$, we obtain the reverse inequality.

Now let us come back to the statement of the theorem. Let $\mathcal{S} = (U,L)$ be a sandwich over A with $U = \{u_1, \ldots, u_n\}$ and $L = \{l_1, \ldots, l_k\}$. Since $\mathcal{S}$ is a sandwich, for every $l_j \in L$ there exists $u_{i_j} \in U$ such that $l_j \uparrow u_{i_j}$. Let $\mathcal{I} \subseteq [n] \times [k]$ be the set of pairs of indices such that $(i,j) \in \mathcal{I} \Leftrightarrow u_i \uparrow l_j$. Then

$$\mathcal{S} = \sum_{(i,j)\in\mathcal{I}} (u_i, l_j) + \Box \sum_{i=1}^{n} (u_i, u_i) \tag{1}$$

From now on we assume that summation over an empty set is the identity for the $+$ operation. It shows that (1) holds even if one of the components of a sandwich is empty.

Using representation (1), define f^+ for an admissible $f : A \uparrow A \to M$ as follows:

$$f^+(S) = \sum_{(i,j)\in\mathcal{I}} f(u_i, l_j) + \Box \sum_{i=1}^{n} f(u_i, u_i) \tag{2}$$

Let us show that f^+ is a homomorphism. Prove that f^+ is monotone first. Let $\mathcal{S}_1 = (U,L)$ and $\mathcal{S}_2 = (V,M)$ be two sandwiches such that $\mathcal{S}_1 \sqsubseteq^{\mathbb{B}} \mathcal{S}_2$, that is, $U \sqsubseteq^{\natural} V$ and $L \sqsubseteq^{\flat} M$. Let $\mathcal{S} = (U,M)$. Observe that $\mathcal{S}$ is a sandwich. Therefore, the proof of $f^+(\mathcal{S}_1) \leq f^+(\mathcal{S}_2)$ is contained in the following two claims.

Claim 1: $f^+(\mathcal{S}_1) \leq f^+(\mathcal{S})$.

Proof of claim 1: If $L = \emptyset$, then claim follows easily from (1), admissibility and equation 4 of mix algebras. For $L \neq \emptyset$, since $L \sqsubseteq^{\flat} M$, there is a sequence of sets $L_0 = L, L_1, \ldots, L_n = M$ such that each $L_i \subseteq L \cup M$ and either $L_{i+1} = \max(L_i \cup l)$ or $L_{i+1} = \max((L_i - L') \cup l)$ where $l' \precsim l$ for all $l' \in L'$, see proposition 3 of [21]. Then each (U, L_i) is a sandwich. We must show $f^+(U,L_i) \leq f^+(U,L_{i+1})$. Consider the first case, i.e. $L_{i+1} = \max(L_i \cup l)$. To verify $f^+(U,L_i) \leq f^+(U,L_{i+1})$ in this case, it is enough to show $\Box f(u,u) + f(u,l) \geq \Box f(u,u)$ if $u \uparrow l$ and, if there is an element $l' \in L$ such that $l' \leq l$, then $f(u',l') + f(u,l) + \Box f(u,u) \geq f(u',l') + \Box f(u,u)$

if $u' \not\upharpoonright l'$. The first one is easy: $\Box f(u,u) + f(u,l) = \Box f(u,l) + f(u,l) = f(u,l) \geq \Box f(u,u)$. The second one follows from monotonicity of $+$: $f(u,l) + \Box f(u,u) \geq \Box f(u,l) = \Box f(u,u)$.

Consider the second case, i.e. $L_{i+1} = \max((L_i - L') \cup l)$. Assume $u \not\upharpoonright l$. Then $u \not\upharpoonright l'$ for any $l' \in L'$. Therefore, any summand $f(u,l)$ in (2) for (U, L_{i+1}) is bigger than $f(u,l')$ in (2) for (U, L_i). Now suppose there is $l' \in L'$ such that $u' \not\upharpoonright l'$ but u' is not consistent with l. If l is consistent with some $u \in U$, then $u \not\upharpoonright l'$. Therefore, to finish the proof of claim 1, we must show that $f(u',l') + f(u,l') \leq f(u,l)$. But this follows from admissibility of f: $f(u',l') + f(u,l) \leq f(u,l') \leq f(u,l)$. Claim 1 is proved.

Claim 2: $f^+(\mathcal{S}) \leq f^+(\mathcal{S}_2)$.

Proof of claim 2: Again, we assume non-emptiness, since for empty sets the proof of claim 2 readily follows from (1). We start with proving the following. Given a sandwich (W,N) and $n \in N$, let w_n be arbitrarily chosen element of W such that $w_n \upharpoonright n$. Then, given an admissible function f, $f^+(W,N)$ defined by (2) equals $\sum_{n \in N} f(w_n, n) + \Box \sum_{w \in W} f(w,w)$. To prove this, assume that there are two elements w_1 and w_2 in W consistent with $n \in N$. Then we must show $f(w_1,n) + f(w_2,n) + \Box f(w_1,w_1) + \Box f(w_2,w_2) = f(w_1,n) + \Box f(w_1,w_1) + \Box f(w_2,w_2)$. That the left hand side is less than the right hand side follows from admissibility. On the other hand, $f(w_1,n) + \Box f(w_1,w_1) + \Box f(w_2,w_2) = f(w_1,n) + \Box f(w_2,n) + \Box f(w_1,w_1) + \Box f(w_2,w_2) \leq f(w_1,n) + f(w_2,n) + \Box f(w_1,w_1) + \Box f(w_2,w_2)$ which proves our claim.

Now, to prove claim 2, consider $\mathcal{S}_2 = (V,M)$ and let v_m be an element of V consistent with $m \in M$. Since $U \sqsubseteq^\sharp V$, let u_m be an element of U under v_m. Then $u_m \upharpoonright m$. Also, let u^v be an element of U under $v \in V$. Then $\Box \sum_{u \in U} f(u,u) = \Box \sum_{v \in V} f(u^v, u^v) + \Box \sum_{u \neq u^v} f(u,u) \leq \Box \sum_{v \in V} f(u^v, u^v) \leq \Box \sum_{v \in V} f(v,v)$. Now, by the claim proved above, $f^+(\mathcal{S}) = \sum_{m \in M} f(u_m, m) + \Box \sum_{u \in U} f(u,u) \leq \sum_{m \in M} f(v_m, m) + \Box \sum_{v \in V} f(v,v) = f^+(\mathcal{S}_2)$ which finishes the proof of claim 2 and monotonicity of f^+.

Now we demonstrate that f^+ preserves the operations of the signature of the mix algebras. Since $\Box$ distributes over $+$, $\Box f^+(\mathcal{S}) = \sum_{(i,j) \in \mathcal{I}} \Box f(u_i, l_j) + \sum_i \Box f(u_i, u_i)$. Since $\Box f(u_i, l_j) + \Box f(u_i, u_i) = \Box f(u_i, u_i)$, we obtain $\Box f^+(\mathcal{S}) = \sum_{i=1}^n \Box f(u_i, u_i) = f^+(\Box \mathcal{S})$. Moreover, since $\Box e = e$, this also holds when one of components is empty. In addition, $f^+(\emptyset, \emptyset) = e$.

That f^+ is a $+$-homomorphism easily follows from (2) when one of the components is empty. So in the rest of the proof we assume that the second components of all sandwiches are not empty.

Let $\mathcal{S}_1 = (U,L)$, $\mathcal{S}_2 = (V,M)$. Let $\mathcal{S} = \mathcal{S}_1 + \mathcal{S}_2 = (W,N)$. Consider a pair (u_i, l_j) with $(i,j) \in \mathcal{I}$. There are three cases: this pair is either present in the representation (1) of $\mathcal{S}$ or $u_i \succsim v_k$ for some $v_k \in V \cap \min(U \cup V)$ or $l_j \precsim m_k \in M \cap \max(L \cup M)$.

Consider the second case. We have $v_k \not\upharpoonright l_j$. Assume $l_j \precsim p$ and $p \in N$. We know that $p \not\upharpoonright q$ for some $q \in W$. Since $f(v_k, l_j) + f(q,p) + \Box f(v_k, v_k) = f(q,p) + \Box f(v,v)$ by 2), we obtain $f^+(\mathcal{S}) = f^+(\mathcal{S}) + f(v_k, l_j)$. Furthermore, since $\Box f(v_k, v_k) + f(u_i, l_j) + f(v_k, l_j) = \Box f(v_k, v_k) + f(v_k, l_j)$ by 1), we have $f^+(\mathcal{S}) = f^+(\mathcal{S}) + f(v_k, l_j) + f(u_i, l_j)$.

Consider the third case. Assume u_i is greater or equal than some $v \in W$ and $m_k \not\upharpoonright q$ for $q \in W$. Then $f(v, l_j) + f(q, m_k) = \Box f(v,v) + f(q, m_k)$ by 3), and hence $f^+(\mathcal{S}) = f^+(\mathcal{S}) + f(v, l_j)$. Since $f(v, l_j) = f(u, l_j) + \Box f(v,v)$ by 4), we obtain $f^+(\mathcal{S}) = f^+(\mathcal{S}) + f(u_i, l_j)$.

Assume that $u \succsim v$. Since $\Box f(u,u) + \Box f(v,v) = \Box f(v,v)$ by 5), we obtain $f^+(\mathcal{S}) = f^+(\mathcal{S}) + \Box f(u_i, u_i)$ for any u_i.

All this shows that $f^+(\mathcal{S})$ can be rewritten as $f^+(\mathcal{S}_1) + f^+(\mathcal{S}_2) + X$ where X is a sum of some elements of form $f(u_i, m_j)$ or $f(v_i, l_j)$. Consider a pair (u_i, m_j) such that $u_i \not\upharpoonright m_j$. There exists v_k such that $v_k \not\upharpoonright m_j$. Since $f(v_k, m_j) + \Box f(u_i, u_i) = f(v_k, m_j) + \Box f(u_i, u_i) + f(u_i, m_j)$ by 6), the summand $f(u_i, m_j)$ can be safely removed from X. Thus, any summand can be removed from X and $f^+(\mathcal{S}) = f^+(\mathcal{S}_1) + f^+(\mathcal{S}_2)$. Therefore, f^+ is a homomorphism.

The uniqueness of f^+ follows from (1). Since $f^+(\eta^\dagger(x,x)) = f(x,x)+\Box f(x,x) = f(x,x)$, we have $f^+ \circ \eta^\dagger = f$. The theorem is proved. □

Next, we must show that the order on bi-LNB algebras is well-defined.

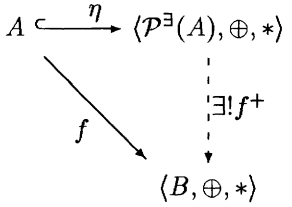
Lemma 1 *In a distributive bi-LNB algebra, $a \leq b := b \oplus a = a * b$ defines a partial order. Moreover, $\oplus$ and $*$ are monotone with respect to $\leq$.*

Proof. First, let us show that $b \oplus a = a * b$ implies $a \oplus b = a$ and $b * a = b$. If $a * b = b \oplus a$, then $b * a = b * a * b = b * (b \oplus a) = b \oplus b * a = b \oplus b = b$. Moreover, $a = a \oplus a = a \oplus a * b = a \oplus b \oplus a = a \oplus b$.

Because of idempotency, $\leq$ is reflexive. To prove transitivity, let $a \leq b$ and $b \leq c$. We must show $a * c = c \oplus a$. Calculate $c \oplus a = c * b \oplus a \oplus b = (c \oplus b) * b \oplus a = b * c * b \oplus a = b * c \oplus a = (b \oplus a) * (c \oplus a) = a * b * c \oplus a * b * a = a * b * c \oplus a = a * b * c \oplus a * b * c = a * b * c$. On the other hand, $a * c = (a \oplus b) * c * b = a * c * b \oplus b * c * b = a * c * b \oplus b = (a \oplus b) * (c \oplus b)b = a * b * c * b = a * b * c$. Hence, $c \oplus a = a * c$ and $a \leq c$. Finally, if $a \leq b$ and $b \leq a$, then $a \oplus b = a$ and $b * a = b$. Hence, $b = b * a = a \oplus b = a$, which finishes the proof that $\leq$ is a partial order.

Assume that $a \leq b$. To see that $a \oplus c \leq b \oplus c$, calculate $(a \oplus c) * (b \oplus c) = a * b \oplus a * c \oplus c * b \oplus c = a * b \oplus a \oplus c = b \oplus a \oplus c = (b \oplus c) \oplus (a \oplus c)$. Similarly, $\oplus$ is monotone in its second argument. To show $a * c \leq b * c$, calculate $a * c \oplus b * c = (a \oplus b) * c = b * a * c = b * c * a * c$. Similarly, $c * a \oplus c * b = c * (a \oplus b) = c * b * a = c * a * c * b$. Hence, $*$ is monotone. □

Proof of theorem 4. We must show that for any distributive bi-LND algebra B and any monotone map $f : A \to B$, there exists a unique homomorphism which completes the following diagram:



First observe that if $(U, L) \in \mathcal{P}^\exists(A)$, then $U, L \neq \emptyset$. We leave it to the reader to find an easy proof that $\mathcal{P}^\exists(A)$ satisfies all equations of the distributive bi-LND algebras under the given interpretation of $\oplus$ and $*$ and that $\mathcal{S}_1 \sqsubseteq^{\mathbb{B}} \mathcal{S}_2$ iff $\mathcal{S}_1 * \mathcal{S}_2 = \mathcal{S}_2 \oplus \mathcal{S}_1$. Given $(U, L) \in \mathcal{P}^\exists(A)$, we can find $u \in U$ and $l \in L$ such that $u_1 \precsim l_1$. Then, using $\sum\!\!\!\!\!\!\sum$ for repeated applications of $\oplus$, and $\bigotimes$ for repeated applications of $*$, we can see that

$$(U, L) = \sum\!\!\!\!\!\!\sum_{u \in U} \eta(u) * \eta(u_1) * \eta(l_1) * \bigotimes_{l \in L} \eta(l)$$

if in the summation over elements of U the first summand is below an element of L. Now, given a monotone f from A into an algebra B, define $f^+ : \mathcal{P}^\exists(A) \to B$ as follows:

$$f^+(U, L) = \sum\!\!\!\!\!\!\sum_{u \in U} f(u) * f(u_1) * f(l_1) * \bigotimes_{l \in L} f(l)$$

Our first goal is to show that in the above representation any number of expressions of form $f(u') * f(l')$, where $u' \precsim l'$, can be added after $f(u_1) * f(l_1)$. This is indeed correct, as $f(u') \leq f(l')$ implies $f(u') * f(l') = f(l')$ and $f(l')$ is subsumed by $\bigotimes_{l \in L} f(l)$.

Denote $f(u_1) \oplus \ldots \oplus f(u_n)$ by $\tilde{U}$ for $U = \{u_1, \ldots, u_n\}$ and $f(l_1) * \ldots * f(l_k)$ by $\hat{L}$ for $L = \{l_1, \ldots, l_k\}$. Then $f^+((U, L)) = \tilde{U} * f(u_{i_1}) * \ldots * f(u_{i_m}) * \hat{L}$ for any number of u_{i_j}'s which are under some elements of L.

To show that f^+ is well-defined, we must prove that its value does not change if we pick a different first summand in $\tilde{U}$ as long as it is below an element of L. It suffices to prove the following. Let $u_i \leq l_i$, $i = 1, 2$. Then $(f(u_1) \oplus f(u_2)) * \hat{L} = (f(u_2) \oplus f(u_1)) * \hat{L}$. This can be further reduced to proving $(f(u_1) \oplus f(u_2)) * f(l_1) * f(l_2) = (f(u_2) \oplus f(u_1)) * f(l_1) * f(l_2)$. Again, we calculate

$$\begin{aligned}(f(u_1) \oplus f(u_2)) * f(l_1) * f(l_2) = f(u_1) * f(l_1) * f(l_2) \oplus f(u_2) * f(l_1) * f(l_2) &= \\ (f(l_1) \oplus f(u_1)) * f(l_2) \oplus (f(l_2) \oplus f(u_2)) * f(l_1) &= \\ f(l_1) * f(l_2) \oplus f(l_2) * f(l_1) \oplus f(u_1) * f(l_2) \oplus f(u_2) * f(l_1)\end{aligned}$$

Similarly,

$$(f(u_2) \oplus f(u_1)) * f(l_1) * f(l_2) = f(l_2) * f(l_1) \oplus f(l_1) * f(l_2) \oplus f(u_1) * f(l_2) \oplus f(u_2) * f(l_1)$$

Now the desired equality follows from the equality $(a * b) \oplus (b * a) = (b * a) \oplus (a * b)$ which is true in all bi-LNB algebras.

Our next goal is to show that any number of nonminimal elements can be added to U and any number of nonmaximal elements can be added to L and that it does not change the value of f^+. That is, writing expressions for f^+ we may disregard min and max operations.

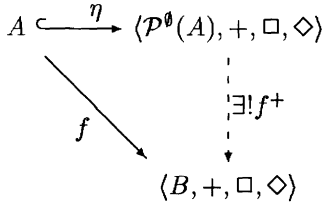
Assume that $u \precsim u'$ and u' is added to U. There are two cases. If $f(u')$ is not the first summand in $\widetilde{U \cup u'}$, then $f(u) \oplus f(u') = f(u)$, so we may disregard $f(u')$. It is also possible that $f(u')$ can be used in the expression for f^+ between $\tilde{U}$ and $\hat{L}$, in which case it can also be disregarded as, if it is below some l, then $f(u') * f(l) = f(l)$. Finally, consider the case when $f(u')$ is the first summand. It is only possible if $u \precsim u' \precsim l$ for some $l \in L$. To prove that $f(u')$ can be dropped and replaced by $f(u)$ in this case, we must show $(f(u') \oplus f(u)) * f(l) = f(u) * f(l)$. Since $f(u) \leq f(u')$ and $f(u') \oplus f(u) = f(u) * f(u')$, we obtain $(f(u') \oplus f(u)) * f(l) = f(u) * f(u') * f(l) = f(u) * f(l) * f(u') = f(u) * f(l)$.

If $l' \precsim l$ is added to L, $f(l')$ does not change the value of f^+ as $f(l) * f(l') = f(l)$. Therefore, we may disregard all max and min operations in expressions for f^+.

At this point we are ready to show that f^+ is a homomorphism. Its uniqueness will follow from the representation of elements of $\mathcal{P}^{\exists}(A)$ from singletons and well-definedness of f^+. Let $\mathcal{S}_1 = (U, L)$ and $\mathcal{S}_2 = (V, M)$. Let $u_1 \precsim l_1$ and $v_1 \precsim m_1$ for $u_1 \in U, l_1 \in L, v_1 \in V, m_1 \in M$. Then $f^+(\mathcal{S}_1) * f^+(\mathcal{S}_2) = \sum_{v \in V}(f^+(\mathcal{S}_1) * f(v) * f(v_1) * \hat{M})$. For two v_i and v_j, consider $f^+(\mathcal{S}_1) * f(v_i) * f(v_1) * \hat{M}$ and $f^+(\mathcal{S}_1) * f(v_j) * f(v_1) * \hat{M}$. Since $L \neq \emptyset$, they are the same, because $a * b \oplus a * c = a * b$ is a derivable equality. Hence, $f^+(\mathcal{S}_1) * f^+(\mathcal{S}_2) = f^+(\mathcal{S}_1) * f(v_1) * \hat{M}$. Since $v_1 \precsim m_1$, we have $f(m_1) * f(v_1) = f(m_1)$ and hence $x * f(v_1) * \hat{M} = x * \hat{M}$ for any x. Thus, $f^+(\mathcal{S}_1) * f^+(\mathcal{S}_2) = \tilde{U} * f(u_1) * \hat{L} * \hat{M} = \tilde{U} * f(u_1) * \widehat{L \cup M} = f^+(\mathcal{S}_1 * \mathcal{S}_2)$. Therefore, f^+ is a $*$-homomorphism.

Now consider $f^+(\mathcal{S}_1) \oplus f^+(\mathcal{S}_2)$. From the equational theory, we immediately have $f^+(\mathcal{S}_1) \oplus f^+(\mathcal{S}_2) = (\tilde{U} * f(u_1) * \hat{L}) \oplus \tilde{V}$. Furthermore, since $(a \oplus c) * b = a * b \oplus c * b = a * b \oplus c$, we have $f^+(\mathcal{S}_1) \oplus f^+(\mathcal{S}_2) = (\tilde{U} \oplus \tilde{V}) * f(u_1) * \hat{L} = \widetilde{U \cup V} * f(u_1) * \hat{L} = f^+(\mathcal{S}_1) \oplus f^+(\mathcal{S}_2)$. Thus, f^+ is a homomorphism. Theorem is proved. □

Proof of theorem 5. We must show that for any bi-mix algebra B and any monotone map $f : A \to B$, there exists a unique homomorphism which completes the following diagram:



It is easy to check that $\mathcal{P}^{\emptyset}(A)$ is a bi-mix algebra under the given interpretation of the operations. To prove freeness, we first need a few facts about bi-mix algebras.

Let $e = \Box\Diamond x$. We have $y + \Diamond x \geq y$ and hence by monotonicity $\Box y + e \geq \Box y$. Adding $\Diamond y$ to both sides, we get by monotonicity that $\Diamond y + \Box y + e \geq \Diamond y + \Box y$ and hence $y \geq y + e \geq y$ which proves that e is the identity of $+$. Similarly, if we define $e' = \Diamond\Box x$, then e' is the identity of $+$ and therefore $e = e'$. This shows that the identity of $+$ can be correctly defined as $e = \Box\Diamond x = \Diamond\Box y$ for arbitrary x and y. Since $x \geq \Box x$, we have $\Diamond x \geq \Diamond\Box x = e$. Similarly, $\Box x \leq e$. It is also easy to see that $\Box e = \Diamond e = e$.

Now, given $(U, L) \in \mathcal{P}^{\emptyset}(A)$, observe that

$$(U, L) \quad = \quad \Box \sum_{u \in U} \eta(u) \; + \; \Diamond \sum_{l \in L} \eta(l)$$

As usual, summation over $\emptyset$ is assumed to be e. Then, given $f : A \to B$, define $f^+ : \mathcal{P}^{\emptyset}(A) \to B$ as follows:

$$f^+((U, L)) \quad = \quad \Box \sum_{u \in U} f(u) \; + \; \Diamond \sum_{l \in L} f(l)$$

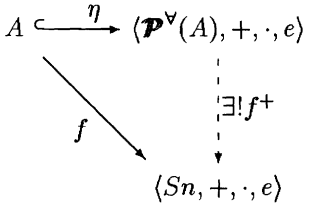
First, $f^+(\eta(x)) = f^+((x, x)) = \Box f(x) + \Diamond f(x) = f(x)$ and hence $f^+ \circ \eta = f$. Now we are going to show that f^+ is a homomorphism. Its uniqueness will then follow from the representation of elements of $\mathcal{P}^{\emptyset}(A)$ given above.

Before we show that f^+ is monotone, let us check that the value of f^+ does not change if an element $l' \precsim l \in L$ is added to L or an element $u' \succsim u \in U$ is added to U. Indeed, to prove the former, observe that $f(l') \leq f(l)$ and $\Diamond f(l') + \Diamond f(l) \leq \Diamond f(l) + \Diamond f(l) = \Diamond f(l)$. For the latter, $\Box f(u) \geq \Box f(u) + \Box f(u') \geq \Box f(u)$ and hence $\Box f(u) + \Box f(u') = \Box f(u)$.

To show that f^+ is monotone, observe that if $(U, L) \sqsubseteq^{\mathbb{B}} (V, M)$, then $U \sqsubseteq_a^{\text{or}} V$ and $L \sqsubseteq_a^{\text{OWA}} M$ and hence V can be obtained from U by a sequence of updates described in proposition 3 of [21] and M can be obtained from L by a sequence of updates described in the same proposition. It is easy to see that updates that replace an element by a number of bigger elements are monotone. Consider removing an element u from U. If $U = \{u\}$, then $\sum_{u' \in \emptyset} \Box f(u') = e \geq \Box f(u)$. If $u' \in U - \{u\}$, then $\Box f(u') \geq \Box f(u') + \Box f(u)$ which proves monotonicity in this case. Finally, if $L = \emptyset$ and an element is l added, then $\Diamond f(l) \geq \sum_{l' \in \emptyset} \Diamond f(l') = e$. If $l \in L$ and l' is added, we have monotonicity because $\Diamond f(l) + \Diamond f(l') \geq \Diamond f(l)$.

Now we are ready to prove that f^+ is preserves $+$, $\Box$ and $\Diamond$. First, $\Box f^+((U, L)) = \Box\Box \sum_{u \in U} f(u) + \Box\Diamond \sum_{l \in L} f(l) = \Box \sum_{u \in U} f(u) + e = f^+(\Box(U, L))$. Similarly, $\Diamond f^+((U, L)) = f^+(\Diamond(U, L))$. The fact that $+$ is preserved follows immediately from the definition of f^+ and the observation that nonminimal elements in U and nonmaximal elements in L do not affect the value of f^+. □

Proof of theorem 6. We have to show that for any snack algebra Sn and a monotone map $f : A \to Sn$, there exists a unique snack homomorphism $f^+ : \boldsymbol{\mathcal{P}}^{\forall}(A) \to Sn$ such that the following diagram commutes:



We omit verification that $\mathcal{P}^{\forall}(A)$ is a snack algebra (in fact, the distributivity laws will be verified later in the greater generality).

Given a snack $\mathcal{S} = (U, \mathcal{L})$ where $U = \{u_1, \ldots, u_n\}$ and $\mathcal{L} = \{L_1, \ldots, L_k\}$, $L_i = \{l^i_1, \ldots, l^i_{k_i}\}$, we have

$$\mathcal{S} = (\prod_{i=1}^{n} \eta(u_i))e + \sum_{i=1}^{k} \prod_{j=1}^{k_i} \eta(l^i_j) \tag{3}$$

Then, if monotone $f : A \to Sn$ is given, define $f^+ : \mathcal{P}^{\forall}(A) \to Sn$ by

$$f^+(\mathcal{S}) = (\prod_{i=1}^{n} f(u_i))e + \sum_{i=1}^{k} \prod_{j=1}^{k_i} f(l^i_j) \tag{4}$$

Clearly, $f^+(\emptyset, \emptyset) = e$ and $f^+(\eta(x)) = f(x) \cdot e + f(x) = f(x)$. We must show that f^+ is a homomorphism.

We start with a few easy observations. First, notice that for a snack algebra $+$ is monotone with respect to $\leq$. Indeed, take $b \geq c$ and observe that $(a+b)(a+c) = a + bc = a + c$, hence $a + b \geq a + c$. Let us now take three elements $a \leq b \leq c$. We have: $ae + c \leq ae + ae + c \leq ae + b + c \leq ae + c + c = ea + c$. Hence, $ae + b + c = ae + c$. Furthermore, consider arbitrary a and b. Since $abe(a+b) = abe$, we have $abe \leq (a+b)e$. On the other hand, $ae + be$ is below a, b and e, and hence $ae + be \leq abe$. Thus, $abe = (a+b)e$.

Let $x \precsim y$ in A. Then $f(x) \leq f(y)$ and hence $f(x) \cdot f(y) = f(x)$. Therefore, if X and Y are two finite subsets of A equivalent with respect to $\sqsubseteq^\sharp$, then $\prod_{x \in X} f(x) = \prod_{y \in Y} f(y)$.

Furthermore, assume $U \sqsubseteq^\sharp X \sqsubseteq^\sharp Y$ for $U, X, Y \in \mathbb{P}_{\text{fin}}(A)$. Then we have $\prod_{u \in U} f(u) \cdot e \leq \prod_{x \in X} f(x) \leq \prod_{y \in Y} f(y)$ and therefore $\prod_{u \in U} f(u) \cdot e + \prod_{x \in X} f(x) + \prod_{y \in Y} f(y) = \prod_{u \in U} f(u) \cdot e + \prod_{y \in Y} f(y)$. This observation shows that writing an expression for $f^+(\mathcal{S}_1 + \mathcal{S}_2)$ and $f^+(\mathcal{S}_1 \cdot \mathcal{S}_2)$ one may disregard all max and min operations. That is, for $\mathcal{S}_1 = (U, \mathcal{L})$ and $\mathcal{S}_2 = (V, \mathcal{M})$,

$$f^+(\mathcal{S}_1 + \mathcal{S}_2) = \prod_{u \in U} f(u) \cdot \prod_{v \in V} f(v) \cdot e + \sum_{L \in \mathcal{L}} \prod_{l \in L} f(l) + \sum_{M \in \mathcal{M}} \prod_{m \in M} f(m) \tag{5}$$

$$f^+(\mathcal{S}_1 \cdot \mathcal{S}_2) = \prod_{u \in U} f(u) \cdot \prod_{v \in V} f(v) \cdot e + \sum_{L \in \mathcal{L}, M \in \mathcal{M}} \prod_{l \in L} f(l) \cdot \prod_{m \in M} f(m) \tag{6}$$

That $f^+(\mathcal{S}_1 + \mathcal{S}_2) = f^+(\mathcal{S}_1) + f^+(\mathcal{S}_2)$ follows immediately from (5).

Let us denote $\prod_{x \in X}$ by $\tilde{X}$. Then $f^+(\mathcal{S}_1 \cdot \mathcal{S}_2) = \tilde{U}\tilde{V}e + \tilde{U}e \cdot \sum_M \tilde{M} + \tilde{V}e \cdot \sum_L \tilde{L} + \sum_L \tilde{L} \cdot \sum_M \tilde{M}$. The last summand is easily seen to be $\sum_{L,M} \tilde{L} \cdot \tilde{M}$. Since $\sum_M \tilde{M} \geq \tilde{V}$, the last summand is also greater than $\tilde{V}e \cdot \sum_L \tilde{L}$ which can therefore be dropped. Similarly, $\tilde{U}e \cdot \sum_M \tilde{M}$ can be dropped. Thus, $f^+(\mathcal{S}_1 \cdot \mathcal{S}_2) = f^+(\mathcal{S}_1) \cdot f^+(\mathcal{S}_2)$ which shows that f^+ is a homomorphism. Its uniqueness follows from (3). □

Proof of theorem 7. Assume that there exists a set of operation Ω_+ such that $\mathcal{P}^{\forall\wedge}(A)$ the free ordered Ω-algebra generated by A for any poset A and $+$ is a derived operation. Let $A = \{x, y, z\}$ be an antichain and $A' = \{x', y', z'\}$ be a poset such that $x', y' \precsim z'$ and $x' \not\precsim y'$, $y' \not\precsim x'$. Let $f : A \to \boldsymbol{\mathcal{P}}^{\forall\wedge}(A')$ be defined by $f(a) = (a', a'), a \in A$. Now the assumed universality property tells us that f can be extended to a monotone Ω_+-homomorphism $f^+ : \boldsymbol{\mathcal{P}}^{\forall\wedge}(A) \to \boldsymbol{\mathcal{P}}^{\forall\wedge}(A')$. Let $\mathcal{S} \in \boldsymbol{\mathcal{P}}^{\forall\wedge}(A')$. Since $\boldsymbol{\mathcal{P}}^{\forall\wedge}(A')$ is the free Ω_+-algebra generated by A', we can find a term t in the signature Ω_+ such that $\mathcal{S} = t(\eta(x'), \eta(y'), \eta(z'))$. Since $\eta(x') = f(x) = f^+(\eta(x))$ and similarly for y' and z', we obtain $\mathcal{S} = f^+(t(\eta(x), \eta(y), \eta(z))) = f^+(\mathcal{S}_0)$ for some $\mathcal{S}_0 \in \boldsymbol{\mathcal{P}}^{\forall\wedge}(A)$. Therefore, f^+ is an onto $+$-homomorphism.

Using the fact that f^+ is a $+$-homomorphism, we find $f^+((xy, \{x, y\})) = f^+((x, x) + (y, y)) = (x', x') + (y', y') = (x'y', \{x', y'\})$ and $f^+((xz, \{x, z\})) = f^+((x, x) + (z, z)) = (x', x') + (z', z') = (x', z')$. Similarly, $f^+((yz, \{y, z\})) = (y', z')$. Define

$$\boldsymbol{\mathcal{P}}_0^{\forall\wedge}(A) = \boldsymbol{\mathcal{P}}^{\forall\wedge}(A) - \downarrow\{(x, x), (y, y), (xy, \{x, y\}), (xz, \{x, z\}), (yz, \{y, z\})\} \quad \text{and}$$

$$\boldsymbol{\mathcal{P}}_0^{\forall\wedge}(A') = \boldsymbol{\mathcal{P}}^{\forall\wedge}(A') - \downarrow\{(x', x'), (y', y'), (x'y', \{x', y'\}), (x', z'), (y', z')\}$$

Since f^+ maps $\boldsymbol{\mathcal{P}}^{\forall\wedge}(A) - \boldsymbol{\mathcal{P}}_0^{\forall\wedge}(A)$ into $\boldsymbol{\mathcal{P}}^{\forall\wedge}(A') - \boldsymbol{\mathcal{P}}_0^{\forall\wedge}(A')$, there must be an onto map from a subset of $\boldsymbol{\mathcal{P}}_0^{\forall\wedge}(A)$ onto $\boldsymbol{\mathcal{P}}_0^{\forall\wedge}(A')$. Now we can find that $\boldsymbol{\mathcal{P}}_0^{\forall\wedge}(A) = \{(xyz, \{x, y, z\}), (z, z), (z, \emptyset)\}$ and $\boldsymbol{\mathcal{P}}_0^{\forall\wedge}(A') = \{(z', z'), (z', \{x', y'\}), (z', x'), (z', y'), (z', x'y'), (z', \emptyset), (x'y', z')\}$. Therefore, there is no map from a subset of $\boldsymbol{\mathcal{P}}_0^{\forall\wedge}(A)$ onto $\boldsymbol{\mathcal{P}}_0^{\forall\wedge}(A')$. This contradiction proves the theorem. □

Proof of theorem 8. Consider two posets: $A = \{x, y, z\}$ and $A' = \{x', y', z'\}$. In A, $x, y \precsim z$ and x and y are incomparable. A' is a chain: $x' \precsim y' \precsim z'$. Define $f : A \to A'$ by $f(x) = x', f(y) = y'$ and $f(z) = z'$. Clearly, f is monotone.

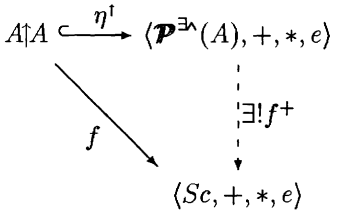
Assume that there exists a signature Ω_+ such that for any poset B, $\langle \boldsymbol{\mathcal{P}}^{\exists}(B), \Omega_+ \rangle$ is the free Ω_+ algebra generated by B. Then we would have a monotone $+$-homomorphism $f^+ : \boldsymbol{\mathcal{P}}^{\exists}(A) \to \boldsymbol{\mathcal{P}}^{\exists}(A')$ such that $f^+((x, x)) = (x', x'), f^+((y, y)) = (y', y')$ and $f^+((z, z)) = (z', z')$. Then we have $f^+((xy, \{x, y\})) = f^+((x, x)+(y, y)) = (x', x')+(y', y') = (x', y')$ and $f^+((y, z)) = f^+((y, y)+(z, z)) = (y', y')+(z', z') = (y', z')$.

Since f^+ is monotone and $(x, xy) \leq (x, x)$, we obtain $f^+((x, xy)) = (x', x')$. Similarly, $f^+((xy, xy)) = (x', x')$. Then $(x', x') = f^+((xy, xy)) = f^+((x, xy) + (y, xy)) = (x', x') + f^+((y, xy))$. Since $(y, xy) \leq (y, y)$, $f^+((y, xy))$ can be either (y', y') or (x', y') or (x', x'). The equality above then tells us that $f^+((y, xy)) = (x', x')$.

Now we use these values of f^+ to calculate $(y', z') = f^+((y, z)) = f^+((y, xy) + (y, z)) = f^+((y, xy)) + f^+((y, z)) = (x', x') + (y', z') = (x', z')$. This contradiction shows that $f : A \to A'$ can not be extended to a monotone $+$-homomorphism between $\boldsymbol{\mathcal{P}}^{\exists}(A)$ and $\boldsymbol{\mathcal{P}}^{\exists}(A')$ and hence $\boldsymbol{\mathcal{P}}^{\exists}(A)$ is not a free Ω_+-algebra generated by A. □

Proof of theorem 9.

Proof of part 1. We must show that for any scone algebra Sc and an admissible map $f : A\!\uparrow\!A \to Sc$, there exists a unique scone homomorphism $f^+ : \boldsymbol{\mathcal{P}}^{\exists\wedge}(A) \to Sc$ which completes the following diagram:



We shall verify the distributivity laws in the proof of algebraic characterization of the salads in the next subsection. Distributivity laws for scones then follow from the observation that the second components of $(U, \mathcal{L}) \cdot (V, \mathcal{M})$ and $(U, \mathcal{L}) * (V, \mathcal{M})$ coincide. Equation 4) is immediate. Thus, $\boldsymbol{\mathcal{P}}^{\exists\wedge}(A)$ is a scone algebra.

We now need some observations about the scone algebras. In what follows, f is an admissible map from $A\restriction A$ to a scone algebra Sc. The definition of admissibility can be rewritten to $f(u,l)*f(v,m) = f(u,l)*f(w,m) = f(u,m) * f(v,l)$.

1) $+$ is monotone with respect to the ordering given by $\cdot$.

Let $b \leq a$. Then $(a+c)(b+c) = (a+c)*(b+c) + (b+c)*(a+c) = c + a*b + b*a = c + ab = b + c$, i.e. $b + c \leq a + c$.

2) $\cdot$ distributes over $+$.

$x(y+z) = x*(y+z) + (y+z)*x = x*y + y*x + x*z + z*x = xy + xz$.

3) If $a \leq b$, then $a*e \leq b*e$.

$(a*e)\cdot(b*e) = a*b*e + b*a*e = (a*b + b*a)*e = (ab)*e = a*e$.

4) $f(x,y) + f(z,y) \leq f(x,y)$.

$f(x,y) + f(z,y) \cdot f(x,y) = (f(x,y) + f(z,y)) * f(x,y) + f(x,y) * (f(x,y) + f(z,y)) = (f(x,y) + f(x,y) * f(z,y)) + f(z,y) * f(x,y) = f(x,y) + f(z,y) * f(z,y) = f(x,y) + f(z,y)$.

5) If $a \precsim b$, then $f(a,a) * e + f(b,b) * e = f(a,a) * e$.

First of all, $f(a,a)*e + f(b,b)*e = f(a,a)*e + f(b,a)*e = (f(a,a) + f(b,a))*e \leq f(a,a)*e$ by 3) and 4). Furthermore, $f(a,a) = f(a,a)+f(a,a) \leq f(a,a)+f(b,b)$ by 1) and therefore $f(a,a)*e \leq (f(a,a)+f(b,b))*e$ which finishes the proof.

6) If $a \precsim b$ and $b\restriction x$, then $f(x,a) * f(b,b) = f(x,a)$.

We have $f(x,a) * f(b,b) = f(x,a) * f(x,b) = f(x,b) * f(x,a)$. Hence $f(x,a) * f(b,b) = f(x,a) * f(x,b) + f(x,b) * f(x,a) = f(x,a) \cdot f(x,b) = f(x,a)$ because $f(x,a) \leq f(x,b)$.

7) For any $a\restriction b$, $f(a,b) * f(b,a) \leq f(a,b)$.

It is easy to see that $(f(a,b) * f(b,a)) \cdot f(a,b) = f(a,b) * f(b,a)$.

8) If $a \precsim b$, then $f(b,b) * f(a,a) = f(b,a)$.

By admissibility and 7), $f(b,b) * f(a,a) = f(b,a) * f(a,b) \leq f(b,a)$. On the other hand, $f(b,a) \cdot (f(b,b) * f(a,a)) = f(b,a)*f(b,b)*f(a,a)+f(b,b)*f(a,a)*f(b,a) = f(b,a)*f(b,b)*f(b,a)+f(b,b)*f(b,a)*f(b,a) = f(b,a) * f(b,b) + f(b,b) * f(b,a) = f(b,a) \cdot f(b,b) = f(b,a)$. Hence, $f(b,a) \leq f(b,b) * f(a,a)$ which proves 8).

Since $\prod$ is already used to denote repeated applications of $\cdot$, for many applications of $*$ we shall use $\otimes$.

Let $\mathcal{S} = (U, \mathcal{L})$ be a scone over A. Since $\uparrow U \cap \uparrow L \neq \emptyset$ for all $L \in \mathcal{L}$, there exists a pair $(u_i, l^j_{k_i})$ for every j

such that $u_i \upharpoonright l^j_{k_i}$. Let $i(j)$ and $k(j)$ be some indices such that $u_{i(j)} \upharpoonright l^j_{k(j)}$. Then $\mathcal{S}$ can be represented as

$$\mathcal{S} = \sum_{u \in U} \eta^\dagger(u,u) * e + \sum_{L_j \in \mathcal{L}} (\eta^\dagger(u_{i(j)}, l^j_{k(j)}) * \bigotimes_{l \in L_j} \eta^\dagger(l,l)) \tag{7}$$

Recall that summation over $\emptyset$ is the identity. We will never need product over the empty index set for all antichains in the second component are nonempty. Moreover, observe that in (7) it does not matter how pairs $(i(j), k(j))$ are chosen.

Using (7), define

$$f^+(\mathcal{S}) = \sum_{u \in U} f(u,u) * e + \sum_{L_j \in \mathcal{L}} (f(u_{i(j)}, l^j_{k(j)}) * \bigotimes_{l \in L_j} f(l,l)) \tag{8}$$

Our first goal is to verify that f^+ is well-defined, that is, it does not depend on how pairs $i(j), k(j)$ are chosen. To save space, denote $\bigotimes_{l \in L} f(l,l)$ by $\hat{L}$. First observe that any number of applications of f to a consistent pair (u,l) for $l \in L_j$ can be put after $f(u_{i(j)}, l^j_{k(j)})$ because, by admissibility, $f(u_{i(j)}, l^j_{k(j)}) * f(u,l) = f(u_{i(j)}, l^j_{k(j)}) * f(l,l)$ and $*$ is idempotent. To finish the proof of well-definedness, it is enough to show that the following equation holds: $f(u,u) * e + f(u',u') * e + f(u,l) * \hat{L} = f(u,u) * e + f(u',u') * e + f(u',l') * \hat{L}$ where $u, u' \in U$ and $l, l' \in L$. By distributivity, this reduces to showing that $f(u,u) * e + f(u',u') * e + f(u,l) * f(l',l') = f(u,u) * e + f(u',u') * e + f(u',l') * f(l,l)$. Because of the symmetry in this equation, it is enough to prove

$$f(u,u) * e + f(u',u') * e + f(u,l) * f(l',l') \leq f(u,u) * e + f(u',u') * e + f(u',l') * f(l,l)$$

Denote $f(u,u) * e + f(u',u') * e$ by p, $f(u,l) * f(l',l')$ by q and $f(u',l') * f(l,l)$ by r. We must show $q + p \leq r + p$. By 2), $(q+p)(r+p) = rq + rp + qp + p$. By monotonicity of $+$ (see 1)), it enough to prove $qp \leq r$. We prove more. In fact, $p \leq r$. First observe that if $a \leq b$, then $a * e \leq b * c$. Indeed, $(a * e) \cdot (b * c) = a * e + b * e = a * e$ by the same argument as in 5). Thus, we must show $p \leq f(u,l)$. Calculate $p \cdot f(u,l) = (f(u,u) + f(u',u')) * e \cdot f(u,l) = (f(u,u) + f(u',u')) * e * f(u,l) + f(u,l) * (f(u,u) + f(u',u')) * e = (f(u,u) + f(u',u')) * e + f(u,l) * e = f(u,u) * e + f(u',u') * e = p$. Thus, $p \leq r$ and this finishes the proof of well-definedness.

Our next goal is to show, as we did for snacks, that if we drop max and min in defining operations on scones, formula (7) will remain true. That will make it much easier to prove that f^+ is a homomorphism.

First observe that if $u \in U$ and $v \gtrsim u$, then $\tilde{U} * e = \widetilde{U \cup v} * e$ (we use notation $\tilde{U}$ as a shorthand for $\sum_{u \in U} f(u,u)$). This follows immediately from 5).

Consider the $\mathcal{L}$-part. In order to show that for $l' \gtrsim l \in L$, the corresponding summand of (8) remains the same if $f(l',l')$ is added, we must show $f(u,l_0) * f(l,l) * f(l',l') = f(u,l_0) * f(l,l)$. The left hand side is equal to $f(u,l_0) * f(l,l) * f(l,l')$ and by 6) $f(l,l) * f(l,l') = f(l,l)$. Therefore, the left hand side is equal to $f(u,l_0) * f(l,l)$.

Finally, it must be shown that adding $M \sqsubseteq^\sharp L \in \mathcal{L}$ does not change the value of the right hand side of (8). Assume $u \in U$, $m \in M$ and $l \in L$ are such that $m \leq l$ and $u \upharpoonright l$ (we can find such because of the consistency condition and $M \sqsubseteq^\sharp L$). Let $a = \hat{L}$ and $b = \hat{M}$. We must show $f(u,l) * a + f(u,m) * b = f(u,l) * a$ (it was already shown that it does not matter which consistent pair is chosen in representation (8)). Let $a' = f(u,l) * a$ and $b' = f(u,m) * b$. First, $a' \cdot b' = (f(u,l) * f(u,m) + f(u,m) * f(u,l)) * a * b = (f(u,l) \cdot f(u,m)) * a * b = f(u,m) * a * b$. Since $L \sqsubseteq^\sharp M$ and $f(c,c) * f(d,d) = f(d,c)$ for $d \gtrsim c$ by 8), we obtain $a' \cdot b' = f(u,m) * b = b'$. Hence $b' \leq a'$ and $a' + b' \leq a'$ by 1). To prove the reverse inequality, $a' \leq a' + b'$, calculate $a' \cdot (a' + b') = a' + (a' \cdot b') = a' + a' * b' + b' * a' = f(u,l) * a + f(u,l) * f(u,m) * a * b + f(u,m) * f(u,l) * a * b$. By admissibility, $f(u,l) * f(u,m) = f(u,m) * f(u,l)$. Therefore, $a' \cdot (a' + b') = f(u,l) * a + f(u,l) * a * f(u,m) * b = a' + a' * b' = a'$.

Thus, $a' \leq a' + b'$ and this finishes the proof that the summand corresponding to $M \sqsubseteq^\sharp L$ can be added to (8).

Now we are ready to prove that f^+ is a homomorphism. First, $f^+(\emptyset, \emptyset) = e * e + e = e$.

Let $\mathcal{S}_1 = (U, \mathcal{L}_1)$ and $\mathcal{S}_2 = (V, \mathcal{M})$. Writing expression (8) for $f^+(\mathcal{S}_1 + \mathcal{S}_2)$ we can use $U \cup V$ as the first component and $\mathcal{L} \cup \mathcal{M}$ as the second. We know that it does not matter how we pick an element from $U \cup V$ to be consistent with some element of a set from $\mathcal{L} \cup \mathcal{M}$. For every $L \in \mathcal{L}$ choose $u_L \in U$ which is consistent with some $l_L \in L$ and similarly for every $M \in \mathcal{M}$ choose $v_M \in V$ which is consistent with some $m_M \in M$. Then we have

$$f^+(\mathcal{S}_1 + \mathcal{S}_2) = \sum_{u \in U \cup V} f(u,u) * e + \sum_{L \in \mathcal{L}} (f(u_L, l_L) * \hat{L}) + \sum_{M \in \mathcal{M}} (f(v_M, m_M) * \hat{M}) = f^+(\mathcal{S}_1) + f^+(\mathcal{S}_2)$$

Clearly, this also holds if either $\mathcal{L}$ or $\mathcal{M}$ or both are empty.

Let $a_L = f(u,l) * \hat{L}$, $c_M = f(v,m) * \hat{M}$ where $u \natural l$, $v \natural m$, $v \in V$, $u \in U$, $l \in L \in \mathcal{L}$ and $m \in M \in \mathcal{M}$. Let $b = \tilde{U} * e$ and $d = \tilde{V} * e$. Then $f^+(\mathcal{S}_1) * f^+(\mathcal{S}_2) = (\sum_{L \in \mathcal{L}}(a_L + b)) * (\sum_{M \in \mathcal{M}}(c_M + d)) = \sum_{L \in \mathcal{L}, M \in \mathcal{M}}(a_L * c_M + a_L * d + b * c_M + b * d)$. Since $d = \tilde{V} * e$, $a_L * d = a_L * e$ and $a_L * c_M + a_L * d = a_L * c_M + a_L * e = a_L * c_M$. Similarly, $b * d = b * e$. Since $b = \tilde{U} * e$, $b = b * e$. Therefore, $b * c_M = b * e = b$ and $b * d = b * e = b$. Therefore, $f^+(\mathcal{S}_1) * f^+(\mathcal{S}_2) = \sum_{L \in \mathcal{L}, M \in \mathcal{M}}(a_L * c_M) + b$. Consider $a_L * c_M$. Since $f(v,m)$ occurs inside the expression, by admissibility it can be changed to $f(m,m)$. Therefore, $a_L * c_M = f(u,l) * \hat{L} * \hat{M}$. Thus,

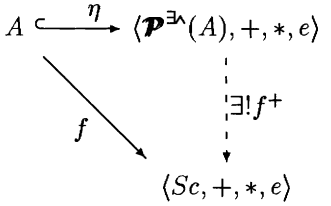
$$f^+(\mathcal{S}_1) * f^+(\mathcal{S}_2) \;=\; b + \sum_{L \in \mathcal{L}, M \in \mathcal{M}} f(u,l) * \hat{L} * \hat{M} \;=$$

$$\sum_{u \in U} f(u,u) * e + \sum_{N \in \{L \cup M \mid L \in \mathcal{L}, M \in \mathcal{M}\}} f(u,l) * \hat{N} \;=\; f^+(\mathcal{S}_1 * \mathcal{S}_2)$$

Now, to finish that proof that f^+ is a homomorphism, it is enough to show that $f^+(\mathcal{S}_1) * f^+(\mathcal{S}_2) = f^+(\mathcal{S}_1 * \mathcal{S}_2)$ if one of the components is empty. Assume $\mathcal{L} = \emptyset$. Then the equation follows from $x * e * y = x * e$ and the fact that $\mathcal{S}_1 * \mathcal{S}_2 = \mathcal{S}_1$. If $\mathcal{M} = \emptyset$, then $f^+(\mathcal{S}_1) * f^+(\mathcal{S}_2) = (\tilde{U} * e + \sum_{L \in \mathcal{L}} f(u_L, l_L) * \hat{L}) * \tilde{V} * e = \tilde{U} * e + \sum_{L \in \mathcal{L}} f(u_L, l_L) * e = \tilde{U} * e = f^+(U, \emptyset) = f^+(\mathcal{S}_1 * \mathcal{S}_2)$. Thus, f^+ is a homomorphism.

The uniqueness of f^+ follows from (7) and well-definedness of (8). Finally, $f^+(\eta^\dagger(x,y)) = f(x,x) * e + f(x,y) * f(y,y) = f(x,y) * e + f(x,y) = f(x,y)$. This shows $f^+ \circ \eta^\dagger = f$. Part 1 is proved.

Proof of part 2. We must prove that for any scone algebra Sc and a scone-admissible map $f : A \to Sc$, there exists a unique scone homomorphism which completes the following diagram:



Let $f : A \to Sc$ be a scone-admissible map. Define $\varphi_f : A \natural A \to Sc$ by

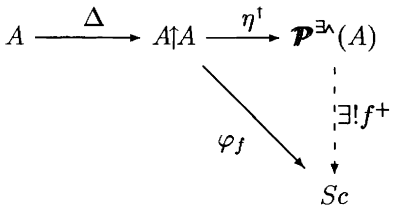
$$\varphi_f((x,y)) = (f(x) * e + f(z)) * f(y) \qquad \text{if } x, y \precsim z$$

It follows from the definition of scone-admissible maps that φ_f is well-defined. That is, if $x, y \precsim z_1, z_2$, then $(f(x)*e+f(z_1))*f(y) = (f(x)*e+f(z_1))*f(y)*f(y) = (f(x)*e+f(z_2))*f(y)*f(y) = (f(x)*e+f(z_1))*f(y)$ and hence the value of $\varphi_f((x,y))$ does not depend on the choice of z above x and y.

Let $\Delta : A \to A \dagger A$ be given by $\Delta(a) = (a, a)$. Our next goal is to prove two claims.

Claim 1. φ_f is admissible (according to definition before this theorem).
Claim 2. $\varphi_f \circ \Delta = f$.

Before we prove these two claims, let us show how the theorem follows from them. Consider the following diagram.



Since φ_f is admissible and $\eta^\dagger \circ \Delta = \eta$, we can find a homomorphism f^+ such that $f^+ \circ \eta = f^+ \circ \eta^\dagger \circ \Delta = \varphi_f \circ \Delta = f$. Assume f^- is another homomorphism $\boldsymbol{\mathcal{P}}^{\exists\wedge}(A) \to Sc$ such that $f^- \circ \eta = f$. Consider $(x, y) \in A \dagger A$, $x, y \precsim z$. Then $\eta^\dagger(x, y) = (\eta(x) * e + \eta(z)) * \eta(y)$. Hence, $f^-(\eta^\dagger(x, y)) = (f(x) * e + f(z)) * f(y) = \varphi_f((x, y))$ which shows that $f^- \circ \eta^\dagger = \varphi_f$. Then, by claim 2 and part 1, we obtain $f^- = f^+$ and thus there is a unique homomorphic extension of f.

Proof of claim 1. First, we must show $\varphi_f((x, y_1)) * e = \varphi_f((x, y_2)) * e$ if $x, y_1 \precsim z_1$ and $x, y_2 \precsim z_2$. From the properties of scone algebras, it follows that $a * e + b * e = a * e$ if $a \leq b$. Since $f(x) \leq f(z_1)$, we obtain $\varphi_f((x, y_1)) * e = (f(x) * e + f(z_1)) * f(y_1) * e = f(x) * e + f(z_1) * e = f(x) * e$. Similarly, $\varphi_f((x, y_2)) * e = f(x) * e = \varphi_f((x, y_1))$.

For the second condition in the definition of admissibility, assume $u, l \precsim x_{ul}$ and $v, m \precsim x_{vm}$. Moreover, let $u, m \precsim x_{um}$ and $w, l \precsim x_{wl}$. We must show $\varphi_f((u, l)) * \varphi_f((v, m)) = \varphi_f((u, m)) * \varphi_f((w, l))$. Observe that $b \geq c$ implies $a * b * c = a * c$ in a scone algebra. Hence, $f(x_{ul}) * f(x_{vm}) * f(m) = f(x_{ul}) * f(m)$. Moreover, as we saw already, $f(u) * e + f(x_{ul}) * e = f(u) * e$. Now we calculate:

$$\begin{aligned} \varphi_f((u, l)) * \varphi_f((v, m)) = (f(u) * e + f(x_{ul})) * f(l) * (f(v) * e + f(x_{vm})) * f(m) &= \\ (f(u) * e + f(x_{ul}) * e + f(x_{ul}) * f(x_{vm})) * f(l) * f(m) &= \\ (f(u) * e + f(x_{ul}) * f(x_{vm})) * f(l) * f(m) = (f(u) * e + f(x_{ul})) * f(l) * f(m) & \end{aligned}$$

Similarly,

$$\varphi_f((u, m)) * \varphi_f((w, l)) = (f(u) + f(x_{um})) * f(l) * f(m)$$

Now the desired equality follows from scone-admissibility of f. Claim 1 is proved.

Proof of claim 2. $\varphi_f((x, x)) = (f(x) * e + f(x)) * f(x) = f(x) * e + f(x) = f(x)$. Claim 2 and part 2 of the theorem are proved.

Proof of part 3. Let Ω_{Sc} be a set of operations on scones such that $+, *$ and e are derived operations. Then $\boldsymbol{\mathcal{P}}^{\exists\wedge}(\cdot)$ is not left adjoint to the forgetful functor from the category of ordered Ω_{Sc}-algebras to **Poset**. In other words, for no Ω_{Sc} is $\boldsymbol{\mathcal{P}}^{\exists\wedge}(A)$ the free ordered Ω_{Sc}-algebra generated by A.

Proof. Let $x, y \precsim z$ in A. Then $((x, x) * (\emptyset, \emptyset) + (z, z)) * (y, y) = (x, y)$. Now consider the following poset $A = \{x, y, z, v\}$. In this poset $x, y \precsim z$, $x, y \precsim v$ and $\{x, y\}$ and $\{z, v\}$ are antichains. Now consider the following scone algebra $Sc_1 = \langle B, +, *, e \rangle$. Its carrier is a four-element chain $p_1 > p_2 > p_3 > p_4$. We interpret $+$ as minimum of two elements, $*$ as maximum, and $e = p_1$. It is easy to see that Sc_1 is a scone algebra as it is a distributive lattice.

Define $f : A \to B$ as follows: $f(z) = p_1, f(v) = p_2, f(x) = p_3$ and $f(y) = p_4$. Now suppose that f can be extended to a homomorphism $f^+ : \mathcal{P}^{\exists\wedge}(A) \to Sc$. Then

$$f^+((x,y)) = f^+((\eta(x) * e + f(z)) * \eta(y)) \quad =$$
$$(f(x) * e + f(z)) * f(y) = \max\{\min\{\max\{p_1, p_3\}, p_1\}, p_4\} = p_1$$

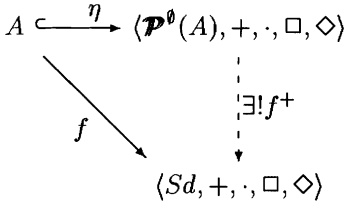
On the other hand,

$$f^+((x,y)) = f^+((\eta(x) * e + f(v)) * \eta(y)) \quad =$$
$$(f(x) * e + f(v)) * f(y) = \max\{\min\{\max\{p_1, p_3\}, p_2\}, p_4\} = p_2$$

Hence, $p_1 = p_2$, which contradicts the definition of B. This shows that f can not be extended to a homomorphism of scone algebras.

Part 3 and theorem 9 are proved. □

Proof of theorem 10. We must show that for every monotone map f from A to a salad algebra Sd there exists a unique salad homomorphism $f^+ : \mathcal{P}^{\emptyset}(A) \to Sd$ such that the following diagram commutes:



First verify that $\mathcal{P}^{\emptyset}(A)$ is a salad algebra. We need to check the distributivity law and 7); all others are straightforward. Let $\mathcal{S}_1 = (U, \mathcal{L}), \mathcal{S}_2 = (V, \mathcal{M})$ and $\mathcal{S}_3 = (W, \mathcal{N})$. Our goal is to show $\mathcal{S}_1 \cdot (\mathcal{S}_2 + \mathcal{S}_3) = \mathcal{S}_1 \cdot \mathcal{S}_2 + \mathcal{S}_1 \cdot \mathcal{S}_3$. The first components of the left hand and the right hand sides coincide. It this case it is easier to work with filters rather than antichains – it allows us to drop max and min operations. In particular, it is enough to show that

$$\{\uparrow(L \cup K) | L \in \mathcal{L}, K \in \mathcal{M} \cup \mathcal{N}\} \quad =$$
$$\{\uparrow L_M | L_M \in \{L \cup M | L \in \mathcal{L}, M \in \mathcal{M}\}\} \bigcup \{\uparrow L_N | L_N \in \{L \cup N | L \in \mathcal{L}, N \in \mathcal{N}\}\}$$

Let C be an element of the left hand side, i.e. $C = \uparrow(L \cup K)$. Without loss of generality, $K \in \mathcal{M}$. Then C is in the right hand side. Conversely, if C is in the right hand side, say $C = \uparrow L_M$ for $L_M = L \cup M$, then $C = \uparrow(L \cup M)$ and therefore is in the left hand side. This shows the equality above. Now, taking minimal elements for each filter and applying $\max^\sharp$ to both collections would give us second components of the lhs and the rhs of the distributivity equation, which therefore are equal.

Now prove 7), that is, $\Diamond(U, \mathcal{L}) \cdot \Diamond(V, \mathcal{M}) + \Diamond(U, \mathcal{L}) = \Diamond(U, \mathcal{L})$. The first components of both sides are $\emptyset$. The second component of the left hand side is $\max^\sharp(\mathcal{L} \cup \max^\sharp\{\min(L \cup M) | L \in \mathcal{L}, M \in \mathcal{M}\})$. Since $\min(L \cup M) \sqsubseteq^\sharp L$, this expression is equal to $\max^\sharp \mathcal{L} = \mathcal{L}$. Hence, 7) holds. Thus, $\mathcal{P}^{\emptyset}(A)$ is a salad algebra.

Now show that $\mathcal{P}^{\emptyset}(A)$ is a free salad algebra. Given a salad $\mathcal{S} = (U, \mathcal{L})$,

$$\mathcal{S} = \Box \sum_{u \in U} \eta(u) + \Diamond \sum_{L \in \mathcal{L}} \prod_{l \in L} \eta(l) \tag{9}$$

To see that this also works for empty components, observe that $\Box e = \Diamond e = e$.

Now, given monotone $f : A \to Sd$, define

$$f^+(\mathcal{S}) = \Box \sum_{u \in U} f(u) + \Diamond \sum_{L \in \mathcal{L}} \prod_{l \in L} f(l) \tag{10}$$

We have: $f^+(\eta(x)) = f^+((x, \{x\})) = \Box f(x) + \Diamond f(x) = x$. Now we must show that f^+ is a homomorphism. First, it follows immediately from the properties of $\Box$ and $\Diamond$ and the fact that $e = \Box\Diamond x = \Diamond\Box y$ is the identity for $+$ (see lemma) that $f^+(\Box\mathcal{S}) = \Box f^+(\mathcal{S})$ and $f^+(\Diamond\mathcal{S}) = \Diamond f^+(\mathcal{S})$.

Assume $X \sqsubseteq^\sharp Y$, $Y \neq \emptyset$, and let x_y be an element in X below $y \in Y$. Then

$$\Box \sum_{x \in X} f(x) \cdot \Box \sum_{y \in Y} f(y) = \Box(\sum_{x \in X} f(x) + \sum_{y \in Y} f(y)) = \Box \sum_{x \in X} f(x) + \Box \sum_{y \in Y} (f(y) + f(x_y)) \quad =$$
$$\Box \sum_{x \in X} f(x) + \Box \sum_{y \in Y} (f(y) \cdot f(x_y)) = \Box \sum_{x \in X} f(x) + \Box \sum_{y \in Y} f(x_y) = \Box \sum_{x \in X} f(x)$$

Therefore, if X and Y are equivalent with respect to $\sqsubseteq^\sharp$, $\Box \sum_{x \in X} f(x) = \Box \sum_{y \in Y} f(y)$. Our next goal is to show that $\Diamond \prod_{x \in X} f(x) + \Diamond \prod_{y \in Y} f(y) = \Diamond \prod_{y \in Y} f(y)$ if $Y \neq \emptyset$. Since $X \sqsubseteq^\sharp Y$, we have $\prod_{x \in X} f(x) \leq \prod_{y \in Y} f(y)$ and then the equation above follows from 7). Finally, let $x' \gtrsim x \in X$. Then $f(x') \geq f(x)$ and $\prod_{x \in X} f(x) = f(x') \cdot \prod_{x \in X} f(x)$.

These three observations show that max and min operations can be disregarded when one writes an expression for f^+ on $\mathcal{S}_1 + \mathcal{S}_2$ or $\mathcal{S}_1 \cdot \mathcal{S}_2$. Therefore, for $\mathcal{S}_1 = (U, \mathcal{L})$ and $\mathcal{S}_2 = (V, \mathcal{M})$,

$$f^+(\mathcal{S}_1 + \mathcal{S}_2) = \Box \sum_{x \in U \cup V} f(x) + \Diamond(\sum_{L \in \mathcal{L}} \prod_{l \in L} f(l) + \sum_{M \in \mathcal{M}} \prod_{m \in M} f(m)) = f^+(\mathcal{S}_1) + f^+(\mathcal{S}_2)$$

To calculate $f^+(\mathcal{S}_1 \cdot \mathcal{S}_2)$, observe that $\sum_{i \in I} \Box x_i \cdot \sum_{j \in J} \Diamond y_j = \sum_{i \in I, j \in J} \Box x_i \cdot \Diamond y_j = \sum_{i \in I} \Box x_i$ and this is also true if $I = \emptyset$ because $e \cdot \Diamond y = e$. Therefore,

$$f^+(\mathcal{S}_1 \cdot \mathcal{S}_2) = (\Box \sum_{u \in U} f(u) + \Diamond \sum_{L \in \mathcal{L}} \prod_{l \in L} f(l)) \cdot (\Box \sum_{v \in V} f(v) + \Diamond \sum_{M \in \mathcal{M}} \prod_{m \in M} f(m)) \quad =$$
$$(\Box \sum_{u \in U} f(u) \cdot \Box \sum_{v \in V} f(v)) + (\Box \sum_{v \in V} f(v) \cdot \Diamond \sum_{M \in \mathcal{M}} \prod_{m \in M} f(m)) \quad +$$
$$+(\Box \sum_{v \in V} f(v) \cdot \Diamond \sum_{L \in \mathcal{L}} \prod_{l \in L} f(l)) + (\Diamond \sum_{L \in \mathcal{L}} \prod_{l \in L} f(l) \cdot \Diamond \sum_{M \in \mathcal{M}} \prod_{m \in M} f(m)) \quad =$$
$$\Box \sum_{u \in U} f(u) + \Box \sum_{v \in V} f(v) + \Diamond \sum_{\substack{L \in \mathcal{L} \\ M \in \mathcal{M}}} (\prod_{l \in L} f(l) \cdot \prod_{m \in M} f(m)) \quad =$$
$$\Box \sum_{x \in U \cup V} f(x) + \Diamond \sum_{\substack{L \in \mathcal{L} \\ M \in \mathcal{M}}} \prod_{y \in L \cup M} f(y) = f^+(\mathcal{S}_1) \cdot f^+(\mathcal{S}_2)$$

Thus, f^+ is a homomorphism. Its uniqueness follows from (9). Theorem is proved. □

Proof sketch of theorem 11. Consider a special case when $f^+[e, u, h]$ is restricted to mixes of form $(U, \emptyset)$ and $h = id$. Then f^+ is equivalent to the structural recursion on sets, whose well-definedness is undecidable, see [4]. The proof for other constructs is similar. □

Proof sketch of theorem 12. Assume monotonicity is decidable. Now, given two $\mathcal{NRL}$ functions $f, g : \{s\} \to t$, define a new function $\phi : \{s\} \to \{bool\}$ as follows:

$$\phi(x) \quad := \quad \textit{if } x = \emptyset \textit{ then } \{true\} \textit{ else if } f(x) = g(x) \textit{ then } \{true\} \textit{ else } \{false\}$$

Now, if want to check whether $f(x) = g(x)$ for all x, check if $f(\emptyset)$ and $g(\emptyset)$ are the same and then check if ϕ is monotone. Thus having a test for monotonicity would give us equality test for functions of type $\{s\} \to t$. Such functions include all functions definable in the relational algebra, and it is known that equality of those is undecidable. This shows that monotonicity of $\mathcal{NRL}$ expressions is undecidable. □

Proof sketch of theorem 13. To prove that the orderings and all operations are definable, observe that $\sqsubseteq^\sharp$ and $\sqsubseteq^\flat$ are first order definable, so they can be defined in the nested relational algebra, and the Buneman orderings are certain combinations of those. It is an easy exercise to see that all operations on approximations are definable that arise from the universality properties.

Moreover, the function that converts all objects into antichain by taking maximal elements for sets and minimal elements for or-sets is also definable in or-$\mathcal{NRL}$. By f_a we shall denote the antichain analog of a function f.

We shall need the following operations from or-$\mathcal{NRL}$. If $f : s \to t$, then $map(f) : \{s\} \to \{t\}$ applies f to all elements of its input. $\eta(x)$ is $\{x\}$. Flattening $\mu : \{\{t\}\} \to \{t\}$ computes union of its arguments. π_1 and π_2 denote the first and second projections. FOr all set operations, there are operations with prefix *or* that act similarly on or-sets.

Now consider mixes. For $f : t \to s\ mix$, where $s\ mix$ is now abbreviation for $\langle s\rangle \times \{s\}$, we have

$$mix\_ext(f) = \lambda(U, L).(or\_\mu_a(or\_map_a(\pi_1 \circ f)(U)), \mu_a(map_a(\pi_2 \circ f)(L)))$$

Mix singleton is defined as $\eta\_mix(x) = (or\_\eta, \eta)$. Then, for $g : s \to t$,

$$mix\_map(g)(U, L) = (or\_map_a(g)(U), map_a(g)(L)) : s\ mix \to t\ mix \qquad \text{and}$$

$$\mu\_mix = \lambda x.(or\_\mu_a(or\_map_a(\pi_1))(x), \mu_a(map_a(\pi_2)(x))) : s\ mix\ mix \to s\ mix$$

As a more complicated example, consider snacks. We use $t\ snack$ as an abbreviation for $\langle t\rangle \times \{\langle t\rangle\}$. important question is how to express $ext\_snack(f) : s\ snack \to t\ snack$ if $f : s \to t\ snack$ is given.

Assume that we have a snack $\mathcal{S} = (U, \mathcal{L})$ of type $s\ snack$. Then $ext\_snack(f)(\mathcal{S})$ can be found as

$$ext\_snack(f)(\mathcal{S}) \quad = \quad (\sum_{u \in U} f(u)) \cdot e + \sum_{L \in \mathcal{L}} \prod_{l \in L} f(l)$$

Look at the first component. If $f(u) = (V_u, N_u)$, then it is equal to $\min(\bigcup_{u \in U} V_u)$ and therefore can be expressed as $C_0 = or\_\mu_a(or\_map_a(\pi_1 \circ f)(\pi_1\ \mathcal{S}))$.

Now fix $L \in \mathcal{L}$. Assume that $f(l) = (W_l, \mathcal{M}_l)$ for each $l \in L$. Then

$$\prod_{l \in L} f(l) = (\min \bigcup_{l \in L} W_l, \max^\sharp(\min(\bigcup_{l \in L} M_l \mid M_l \in \mathcal{M}_l)))$$

To find the first component, compute $or\_\mu_a(or\_map_a(\pi_1 \circ f)(L)$. To find the second component, observe that $X = or\_map_a(\pi_2 \circ f)(L)$ is $\langle \mathcal{M}_l \mid l \in L\rangle$. Therefore, the second component is simply $map_a(or\_\mu_a(\beta_a(X)))$. Here β_a is the inverse of α_a, that is, isomorphism between the semantic domains of types $\langle\{t\}\rangle$ and $\{\langle t\rangle\}$. It is not hard to see that in the presence of *set_to_or* and *or_to_set* it is possible to express β_a in or-$\mathcal{NRL}$.

(We tacitly assumed that these two functions are present. This is the assumption made in [11] but not [21].) Hence, we can write a function

$$g \quad := \quad (or\_\mu_a \circ or\_map_a \circ (\pi_1 \circ f), map_a \circ or\_\mu_a \circ \beta_a \circ or\_map_a \circ (\pi_2 \circ f))$$

which, when applied to L, produces $\prod_{l \in L} f(l) = (Z_L, \mathcal{N}_L)$.

Now we need to calculate $\sum_L (Z_L, \mathcal{N}_L) = (\min \bigcup_L Z_L, \max^\sharp (\bigcup_L \mathcal{N}_L))$. The second component can be obtained as

$$C_2 = \mu_a(map_a(\pi_2 \circ g)(\mathcal{L}))$$

and it is of type $\{\langle t \rangle\}$. To compute the first component, we need a way out of sets to get an or-set. This is achieved by writing $C_1 = or\_\mu_a(set\_to\_or(map_a(\pi_1 \circ g)))(\mathcal{L})$. Finally, we have

$$ext\_snack(f)(\mathcal{S}) = (or\_\cup_a(C_0, C_1), C_2)$$

The proof for other constructions is similar.