Models of Approximation in Databases

Leonid Libkin
Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, USA
email: `libkin@bell-labs.com`

Abstract

Partial information in databases can arise when information from several databases is combined. 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.

The main goal of the paper is to study several formal models of approximations and their semantics. In particular, we obtain universality properties for these models of approximations. Universality properties suggest syntax for languages with approximations based on the operations which are *naturally* associated with them. We prove universality properties for most of the approximation constructs. Then we design languages built around datatypes given by the approximation constructs. A straightforward approach results in languages that 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 [25], and demonstrate its usefulness in querying independent databases. We also discuss the semantics of approximation constructs and relationship between them.

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 [27, 28] suggests the use of two approximations to answer queries Q for which a precise answer cannot be found. The *lower approximation* to Q consists of those objects for which one can conclude with certainty that they belong to the answer to Q. The *upper approximation* to Q consists of those objects for 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 the results of query evaluation, but may also be used as a representation mechanism for certain kinds of partial data. Moreover, this kind of partiality is different from traditional models such as null values and disjunctive information. If a query is asked against several databases, the combined database 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 (or in addition to) incompleteness of the stored data as in the classical models. Let us give some examples.

Example: Querying independent databases

Simple approximations. Consider the following problem. Assume that we have access to two relations in a university database. These relations, Employees and CS1 (for teaching the course CS1), are shown below.

Employees:

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

CS1:

Name	Salary	Room
John	$\perp$	76
Michael	$\perp$	320

Assume that our query asks to compute the set TA of teaching assistants. Suppose that only TAs can teach CS1 and that every TA is a university employee. To make the example easier to understand, we make an assumption that the Name field is a key. We use nulls $\perp$ to make both relations have the same set of attributes. Let us outline how the TA query can be answered. Since every person in CS1 is a TA, CS1 gives us the certain part of the answer. Moreover, every TA is an employee, hence finding people in Employees who are not represented in CS1 gives us the possible part of the answer to the TA query.

The pair of relations CS1 and Employees is called a *sandwich* (for TA), cf. [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 TA – something that's in between; hence the name. Notice that the records in CS1 and Employees are *consistent:* for every record in CS1, there is a record in Employees consistent with it. That is, they are joinable (in the sense of [6, 35]) and their join can be defined. For example,

$$\boxed{\text{John} \mid 15\text{K} \mid \perp} \vee \boxed{\text{John} \mid \perp \mid 76} = \boxed{\text{John} \mid 15\text{K} \mid 76}$$

Note that taking this join makes sense only under the assumption that Name is a key.

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

Assume that a pair of consistent relations U and L is given. What is the semantics of the sandwich (U, L)? That is, what is the family of possible answers to Q that U and L approximate? To answer this question, we appeal to the idea of representing partial objects as elements of ordered sets. In a graphical representation, 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

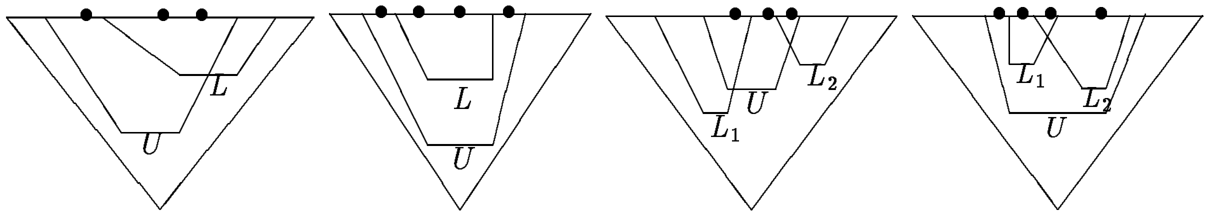

Figure 1: Models of approximations and their semantics

"being less partial," or "being more informative". Maximal elements correspond to complete descriptions, i.e. those that do not have any partial information at all.

For the graphical representation of sets, we depict each set X as a segment, together with all the elements that are above one element of X. In figure 1, each set X is thus shown as a trapezoid "standing" on the segment representing the elements of X.

The picture of a sandwich (U, L) is the leftmost one in figure 1. The semantics of a sandwich is a family of sets such as the one denoted by three bullets in the picture. Such sets X satisfy two properties:

- Every element l of the lower approximation L *approximates* an element of X. That is, for every element $l \in L$, there is an element $x \in X$ such that $l \leq x$.
- Every element x of X is *approximated* by an element of the upper approximation U. That is, for every $x \in X$, there exists $u \in U$ such that $u \leq x$.

Note that in the example shown in figure 1, L is assumed to have two elements, each of them being under an element shown as a bullet. Elements shown as bullets are in turn above some elements of U. Therefore, (U, L) satisfies the consistency condition, i.e. it is a sandwich.

Under the assumption that the Name field is a key, one 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 76 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 at least as informative as some record in the upper approximation. Such a pair is called a *mix*. An example of a mix is shown in figure 1. Mixes were introduced in [13] as an alternative approximation construct, whose properties are generally easier to study than properties of sandwiches because of its simpler consistency condition in which no joins are involved.

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, the set shown by four bullets is an example.

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 TA for that section. Suppose that the selection of TAs has been made, and those selected have been entered in the database of employees, while the database of prospective TAs remained unchanged. This situation is 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	$\perp$	76
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 corresponding 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*, cf. [31]. 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 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 in the lower approximation, one of its elements approximates an element of X. That is, for every set $L \in \mathcal{L}$, there exists $l \in L$ and $x \in X$ such that $l \leq x$. Second, every element of X is approximated by some element of the upper approximation. That is, X lies in the trapezoid standing on U; or, for every $x \in X$, there exists $u \in U$ such that $u \leq x$. An example from Figure 1 is the set denoted by three bullets. Observe that the second property is exactly the same for scones as it is for sandwiches and mixes, while the first one reflects the difference in the structure of scones and sandwiches.

Now let us look at the data represented by $CS1_1$ and $CS1_2$. Assuming that the Name field is a key, one can do some preprocessing before any queries are asked. There is no entry for Jim in Employees. 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 infer some of the null fields as we did before in order to obtain a mix from a sandwich. In the new approximation that we obtain, the condition expressing consistency is much stronger than the condition used for scones. In fact, all elements in $\mathrm{CS1}_1$ and $\mathrm{CS1}_2$ have become elements of Employees. 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 at least as informative as some element of the upper approximation. Such constructions are called *snacks* [29, 31, 19]. The reason for this name is that they were initially thought of – not quite correctly, as we shall show – as "many sandwiches," hence snacks.

The graphical representation of a snack with a two-element $\mathcal{L}$ is given in Figure 1. The semantics of snacks is defined precisely as the semantics of scones. For example, in Figure 1 the four-bullet set 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 violate various consistency conditions. We need a model that does not require any consistency condition at all. Such a model was first introduced in [22]. Since it is in essence "all others put together," it is called a *salad.*

One may ask why we consider lower approximations given by a family of sets, while all upper approximations are just sets. The reason is simple: if upper approximations were allowed to be families of sets, then taking the union of all the elements in the family we would obtain an equivalent approximation. For example, assume that a generalized sandwich of the form $(\{U_1, U_2\}, L)$ is now permitted. The semantics of such a sandwich is the family of all sets X that are approximated by L from below, and such that each element in X is above either an element of U_1 or an element of U_2. But this is the same as the sandwich $(U_1 \cup U_2, L)$. Henceforth, the upper approximation is always a single set.

Goals of the paper and organization. The main problem that we address in this paper is *building the general theory of approximate answers to queries.* In particular, we want to make approximate answers first class citizen objects in a query language. Towards that goal, we focus on 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 correspond to (a combination of) known datatypes?
- How can we program with approximations?

Note that the problems of approximation have been studied by the datalog community; see, for example, [10, 11]. There are, however, major differences between the problems that are addressed. In papers like [10, 11] information is complete, and using approximations reduces the complexity of query evaluation. For example, upper and lower envelopes are defined as datalog programs whose result would always be superset (subset) of a given program P. If P is a recursive program, envelopes are usually sought in the class of conjunctive queries. Secondly, approximating relations are usually defined as subset or superset.

In our approach the reason for approximating is incompleteness of information. Approximations arise as the best possible answers to queries that one can get, and not as the best answers that can be computed within a given complexity class. Moreover, our notions of approximations are much more sophisticated than simple subsets and supersets.

The paper is organized in follows. In section 2 we present preliminary results necessary to describe our approach. First we explain an 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, 18, 21]. One of its important features is that it allows one to abstract from a concrete data model (e.g. relational, complex object) as it can be used with a variety of models [6, 21]. Then we explain a *"data-oriented"* paradigm for query language design [9]. This approach is based on incorporating operations *naturally* associated with datatypes into a query language [8]. 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 classify them.

The main part of the paper is section 4 in which we show that most of the constructs possess universality properties. This tells us what are the important operations on approximations. Obtaining universality properties is an easy task for most datatypes (such as sets, bags, and lists). However, here we encounter a novel situation in which obtaining these properties is difficult. Moreover, we obtain results of a new kind, saying that some constructs do not possess universality properties.

In section 5 we discuss programming with approximation. We apply the data-oriented paradigm to descriptions of approximations obtained in section 4 and discuss problems with using this approach. One problem is the undecidability of certain preconditions that need to be checked to ensure well-definedness of programs. As a solution to this problem, we suggest an encoding of approximation constructs with *or-sets* [17, 25, 33] and explain how the language for or-sets [25] is suitable for programming with approximations. In fact, a system based on this language [15] 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 [3, 16, 12]. In A general approach to the treatment of partial information in the context of ordered sets is developed in [6, 21, 25]. 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 an approach with 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 [3, 6, 22].

Complex objects, or nested relations, are constructed from the base objects by using the record and the set type constructors. Hence, one has to lift an order to records and sets. Lifting to records is done componentwise. For example, [Name: Joe, Age:$\perp$] $\leq$ [Name: Joe, Age: 28].

But 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 [14]. Here we consider two approaches, 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* $\sqsubseteq^\flat$ (generalized subset) or the *Smyth ordering* $\sqsubseteq^\sharp$ (generalized superset):

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

Earlier work on representing partiality via orders did not consider the problem of choosing the right ordering. Recently, a theory for deciding which order is suitable for which collection was developed [25, 22]. It turns out that $\sqsubseteq^\flat$ is suitable for sets[1] and $\sqsubseteq^\sharp$ is suitable for or-sets [17]. 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 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. 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, this ordered semantics suggests that the database objects are represented as antichains in certain posets, cf. [6, 21].

2.2 Data-oriented programming

In this subsection we give an overview of the *data-orientation* as a paradigm for programming language design (cf. Cardelli [9]) and demonstrate one instance of this approach: a language for sets.

It was observed in [9] that while traditional programming languages are mostly algorithmic and procedure-oriented, database languages require more emphasis on data. Databases are designed 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*. This 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 can 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 objects, types are given by the grammar $t ::= b \mid [l_1 : t, \ldots, l_n : t] \mid \{t\}$, where b ranges over base types. We often use *pair* types which are a special case of records: instances of type $t \times s$ are pairs (x, y) where x has type t and y has type s.

[1]Technically speaking, this is true only if we believe in the open world assumption. For closed worlds, the Plotkin ordering [14] 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.

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

Figure 2: Structural recursion and *ext*

It was suggested in [9] 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 for doing computations over them. For example, record formation is the introduction operation for records, and projections are the elimination operations.

Databases work with various kinds of collections. One approach (cf. [8, 4]) to find the introduction and elimination operations for those collections is to look for operations *naturally* associated with them. To do so, one often characterizes the semantic domains of collection types via *universality properties*, which suggest what the introduction and the elimination operations are.

Assume that we have a collection type constructor (like sets, bags etc.) that we denote by $\mathsf{C}(\cdot)$ and a type t. Let $\llbracket t \rrbracket$ denote the semantic domain of type t and $\llbracket \mathsf{C}(t) \rrbracket$ denote the semantic domain of type $\mathsf{C}(t)$ of collections of elements of type t. By *universality property* we mean that the following is true about $\llbracket t \rrbracket$ and $\llbracket \mathsf{C}(t) \rrbracket$. It is possible to find a set Ω of operations on $\llbracket \mathsf{C}(t) \rrbracket$ and a map $\eta : \llbracket t \rrbracket \to \llbracket \mathsf{C}(t) \rrbracket$ such that for any other Ω-algebra $\langle X, \Omega \rangle$ and a map $f : \llbracket t \rrbracket \to X$ there exists a unique Ω-homomorphism f^+ such that the first diagram in figure 2 commutes.

If we are successful in identifying η and Ω, then we can make them the *introduction* operations. The reason is that now any object of type $\mathsf{C}(t)$ can be constructed from objects of type t by first embedding them into type $\mathsf{C}(t)$ by means of η, and then constructing more complex objects using the operations from Ω.

The *elimination* operation is given by the universality property. In fact, the general elimination operation is a higher-order operation that takes f as an input and returns 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, $\mathbb{P}_{\mathrm{fin}}(\llbracket t \rrbracket)$. For any set X, its finite powerset $\mathbb{P}_{\mathrm{fin}}(X)$ is the free semilattice generated by X. That is, the operations of Ω are $\emptyset$ and $\cup$ and η is the singleton formation: $\eta(x) = \{x\}$. We consider $\cup$ and η as *polymorphic operations*: $\cup$ has type $\{t\} \times \{t\} \to \{t\}$ and η has type $t \to \{t\}$.

To include the elimination operation f^+ in a language, we must specify a constant and a binary operation that play the role of the operations of Ω on the range of f^+. That is, f^+ is in fact a parameterized family of functions. Assume that e plays the role of the constant and u plays the role of the binary operation. The operation that takes f into f^+ is the following

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

This operation f^+ is often called *structural recursion* [8]. Notice that if we include it as a query language primitive, there is no guarantee that e and u will satisfy the same equations as $\emptyset$ and $\cup$. But if e and u do not supply the range of f^+ with the structure of a semilattice, then f^+ may not be well-defined. For example, if e is 0, f is $\lambda x.1$, and u is $+$, then $f^+[e,u](\{1\}) = f^+[e,u](\{1\} \cup \{1\})$, thus implying $1 = 2$.

To overcome this problem, originally noticed in [4], one can require that e be interpreted as $\emptyset$ and u as $\cup$. Generally, the simplest way to ensure well-definedness of f^+ 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 $\mathit{ext}(f)$, the extension of f. Its semantics in the case of sets is $\mathit{ext}(f)\{x_1, \ldots, x_n\} = f(x_1) \cup \ldots \cup f(x_n)$ (that is, it "extends" f to sets.) This function is well-defined. Using *ext* together with η, $\emptyset$, $\cup$, projections and record formation, conditional and the equality test gives us precisely the nested relational algebra [8] but the presentation is nicer than the standard ones, such as in [34]. This approach to the language design has proved extremely fruitful in the solution of some open problems (e.g. [26]) and the development of languages for other collection types (e.g. [25, 24]). In order to apply it to the approximation constructs, we need formal models of them as well as the universality properties for these models.

The diagrams above are constructions well known in universal algebra and category theory. The first one says that $[\![\mathsf{C}(t)]\!]$ is the free Ω-algebra generated by $[\![t]\!]$, or, equivalently, 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* [2]. Using monads as the basis for the query language design has been advocated in [8, 7]. The languages thus obtained come equipped with an equational theory, and also admit an easy-to-use comprehension syntax [7].

3 Formal models of approximations

In this section we reexamine the approximation constructs by applying the idea of representing database objects with partial information as elements of certain ordered sets. By giving their formal models, we make it possible to elevate the intuitive notion of approximate answers to first class citizens in a query language. Further towards that goal, we show that the approximation constructs are instances of partial information themselves, and as such can be ordered. We also discuss the formal semantics of the approximation constructs.

We shall 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, consistent means joinable (as in [35].) We shall use $\uparrow X$ for $\{y \mid y \geq x, \text{ some } x \in X\}$ and $\downarrow X$ for $\{y \mid y \leq x, \text{ some } x \in X\}$. We shall call $\uparrow X$ and $\downarrow X$ *filters* and *ideals* (*principal*, if X is a singleton).

3.1 Definition and classification of approximations

Recall that a *sandwich* 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 an $l \in L$ such that u and l are consistent. Representing objects in approximating sets as elements of some posets, we can formally define sandwiches:

Definition 1 (cf. [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: $\forall l \in L\ \exists u \in U : u \uparrow l$ (i.e. $\exists X : L \sqsubseteq^{\flat} X, U \sqsubseteq^{\sharp} X$). The set U is usually referred to as the upper approximation and L as the lower approximation. The family of all sandwiches over A is denoted by $\mathcal{P}^{\forall\wedge}(A)$ (the reason for this notation will be seen shortly).*

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 obtain

Definition 2 (cf. [13].) *Given a poset $\langle A, \leq \rangle$, a* mix *over A is a pair of finite antichains (U, L) satisfying the following consistency condition: $\forall l \in L\ \exists u \in U : u \leq l$ (i.e. $U \sqsubseteq^{\sharp} L$.) The family of all mixes over A is denoted by $\mathcal{P}^{\forall}(A)$.*

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. Hence

Definition 3 (cf. [31].) *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 $i \neq j$. Scones satisfy the consistency condition: $\forall L \in \mathcal{L}\ \exists l \in L\ \exists u \in u : u \uparrow l$ (i.e. $\forall L \in \mathcal{L} : \uparrow L \cap \uparrow U \neq \emptyset$). The family of all scones over A is denoted by $\boldsymbol{\mathcal{P}}^{\exists\wedge}(A)$.*

Snacks are obtained from scones exactly 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 (cf. [29, 31, 19].) *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: $\forall L \in \mathcal{L}\ \forall l \in L\ \exists u \in u : u \leq l$ (i.e. $\forall L \in \mathcal{L} : U \sqsubseteq^{\sharp} L$). The family of all snacks over A is denoted by $\boldsymbol{\mathcal{P}}^{\forall}(A)$.*

Now let us look at these constructs again. One can see that 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

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

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 $u \uparrow l$.

Thus, we have eight constructions since each of the parameters – the structure of the lower approximation, the quantifier $\mathbf{Q}$ and the condition $C(u,l)$ – has two possible values. For constructs with a simple lower approximation we use notation $\mathcal{P}$, for constructs with multi-set lower approximation we use $\boldsymbol{\mathcal{P}}$. The rest is indicated in the superscript whose first symbol is the quantifier $\mathbf{Q}$, that is, $\forall$ or $\exists$. If the condition is $u \uparrow l$, then the second symbol in the superscript is $\wedge$ (to indicate that there is an element above u and l); otherwise, if $C(u,l)$ is $u \leq l$, no second symbol is used. We have seen the need for constructs with no consistency condition, in order to deal with inconsistencies. For two such constructs we shall use just one superscript $\emptyset$.

Summing up, we have ten possible constructs, which are shown in the table below. For example, we denote the family of sandwiches over A by $\mathcal{P}^{\forall\wedge}(A)$, mixes by $\mathcal{P}^{\forall}(A)$, snacks by $\boldsymbol{\mathcal{P}}^{\forall}(A)$ etc.

L-part	*type of consistency condition (quantifier–condition)*				
	$\forall \quad u \leq l$	$\forall \quad u \uparrow l$	$\exists \quad u \leq l$	$\exists \quad u \uparrow l$	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)	$\boldsymbol{\mathcal{P}}^{\emptyset}$

3.2 Ordering approximations

We introduce two orderings $\sqsubseteq^{\natural}$ and $\sqsubseteq^{\natural}_f$ on the approximation constructs. The ordering $\sqsubseteq^{\natural}$ is used for the constructs with a single set in the lower approximation (those denoted by $\mathcal{P}$) and $\sqsubseteq^{\natural}_f$ is used for the constructs with a family of sets in the lower approximation (denoted by $\boldsymbol{\mathcal{P}}$). These are called *the Buneman orderings* [6, 13] and are defined as follows:

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

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

Compactly, $\sqsubseteq^{\natural} = \sqsubseteq^{\sharp} \times \sqsubseteq^{\flat}$ and $\sqsubseteq^{\natural}_f = \sqsubseteq^{\sharp} \times (\sqsubseteq^{\sharp})^{\flat}$. The index f in $\sqsubseteq^{\natural}_f$ indicates that the ordering deals with families of sets in the lower approximations.

Claim. *The approximations are ordered by the Buneman orderings.* □

We refer the reader to [22] for the rationale behind this claim. It is justified by proving the results similar to those proved in [25, 22, 24] for sets, or-sets and bags. That is, a family of elementary transformations is introduced, such that each transformation makes the approximation more precise. Then it is shown that $\sqsubseteq^{\natural}$ and $\sqsubseteq^{\natural}_f$ correspond to the transitive closure of such transformations. We also notice that the Buneman orderings were used in [5, 13].

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 $\sqsubseteq^{\natural}$ and $\sqsubseteq^{\natural}_f$ respectively.

The approximation constructs are similar to (and, in fact, motivated by) the powerdomain constructions used extensively in programming language theory, cf. [14]. We can turn each of the approximation constructs $\mathcal{P}$ into a powerdomain as follows. Given a domain D, apply $\mathcal{P}$ to the poset of the compact elements of D, and take the ideal completion of the result. Several papers [6, 31, 19] adopt this approach and work with powerdomains. We do not believe that using powerdomains is justified in the present context, as the ideal completion helps us model recursive datatypes, and we do not use recursive datatypes in this paper. However, should this become necessary, all the results that follow can easily be generalized to powerdomains, along the line of [13].

3.3 Semantics of approximations

To understand the semantics of the approximation constructs, we use the example from the introduction. For sandwiches and mixes, we assumed that a set TA is approximated by Employees and CS1 if every record in CS1 represents (is less than) a record in TA and every record in TA is represented by (is greater than) a record in Employees. In other words, CS1 $\sqsubseteq^\flat$ TA and TA $\sqsubseteq^\sharp$ Employees.

For scones and snacks, where CS1 was subdivided into a family of relations CS1$_i$, we assumed that at least one element from each CS1$_i$ represents an element in TA. That is, TA $\sqsubseteq^\sharp$ Employees, and for all i, there exists an element in CS1$_i$ that represents an element of TA. In other words, $\uparrow$CS1$_i \cap \uparrow$TA $\neq \emptyset$.

To formalize this, we introduce two semantic functions for the constructs with one- and multi-element lower approximations:

$$[\![(U, L)]\!] = \{X \in \mathbb{P}_{\text{fin}}(A) \mid U \sqsubseteq^\sharp X \text{ and } \quad L \sqsubseteq^\flat X\}$$

$$[\![(U, \mathcal{L})]\!] = \{X \in \mathbb{P}_{\text{fin}}(A) \mid U \sqsubseteq^\sharp X \text{ and } \quad \forall i : \uparrow L_i \cap X \neq \emptyset\}$$

The semantics of mixes and sandwiches has been studied in [5] and [13]. Here we concentrate on the constructs with the multi-element L-part.

Proposition 1 (see also [29]) *If $\mathcal{S}_1$ and $\mathcal{S}_2$ are two snacks, then $\mathcal{S}_1 \sqsubseteq^{\natural}_f \mathcal{S}_2$ iff $[\![\mathcal{S}_2]\!] \subseteq [\![\mathcal{S}_1]\!]$.*

Proof. Let $\mathcal{S}_1 = (U, \mathcal{L})$ and $\mathcal{S}_2 = (V, \mathcal{M})$. Prove the *'if'* part first. Assume $[\![\mathcal{S}_2]\!] \subseteq [\![\mathcal{S}_1]\!]$. Pick arbitrarily an element m_M from each $M \in \mathcal{M}$. Then $V' = V \cup \{m_M | M \in \mathcal{M}\} \in [\![\mathcal{S}_2]\!]$ and therefore $V' \in [\![\mathcal{S}_1]\!]$ which means $U \sqsubseteq^\sharp V' \sqsubseteq^\sharp V$. Hence, $U \sqsubseteq^\sharp V$.

If $\mathcal{M} = \emptyset$, then $\mathcal{L} = \emptyset$ because otherwise $\emptyset \in [\![\mathcal{S}_2]\!]$ but $\emptyset \notin [\![\mathcal{S}_1]\!]$. Hence, in this case $\mathcal{S}_1 \sqsubseteq^{\natural}_f \mathcal{S}_2$. Assume $\mathcal{M} \neq \emptyset$ and $\mathcal{S}_1 \not\sqsubseteq^{\natural}_f \mathcal{S}_2$; then $\exists L \, \forall M \, \exists m \in M \, \forall l \in L : l \not\leq m$. Let $L \in \mathcal{L}$ be a set for which the statement above is true; then, selecting appropriate m for each $M \in \mathcal{M}$ we obtain a set Q such that $Q \cap M \neq \emptyset$ for all $M \in \mathcal{M}$ and $\forall l \in L \forall q \in Q : l \not\leq q$. In other words, $\uparrow L \cap Q = \emptyset$. On the other hand, $Q \in [\![\mathcal{S}_2]\!] \subseteq [\![\mathcal{S}_1]\!]$ and therefore $\uparrow L \cap Q \neq \emptyset$ for all $L \in \mathcal{L}$. This contradiction shows $\mathcal{S}_1 \sqsubseteq^{\natural}_f \mathcal{S}_2$.

To show the *'only if'* part, assume $\mathcal{S}_1 \sqsubseteq^{\natural}_f \mathcal{S}_2$ and $Q \in [\![\mathcal{S}_2]\!]$. Then $U \sqsubseteq^\sharp V \sqsubseteq^\sharp Q$ and, given $L \in \mathcal{L}$, there exist $M \in \mathcal{M}$ such that $\uparrow M \subseteq \uparrow L$ and therefore $Q \cap \uparrow L \neq \emptyset$. Thus $Q \in [\![\mathcal{S}_1]\!]$. $\Box$

Unfortunately, this is no longer true for scones. If $A = \{\bot, \top, a, b, c\}$ is a poset with $\bot$ and $\top$ the bottom and the top elements, and $\{a, b, c\}$ being incomparable, then for two scones $\mathcal{S}_1 = (a, \{b\})$ and $\mathcal{S}_2 = (a, \{c\})$ we have $\{\{\top\}, \{a, \top\}\} = [\![\mathcal{S}_1]\!] = [\![\mathcal{S}_2]\!]$, but $\mathcal{S}_1$ and $\mathcal{S}_2$ are incomparable.

However, there is a very close connection between the semantics of scones and snacks and their ordering. In some sense, the family of snacks over A is the maximal subclass of scones over A on which the semantics and the orderings agree. To formulate this rigorously, let $\mathcal{S}_1 \preccurlyeq \mathcal{S}_2$ iff $[\![\mathcal{S}_2]\!] \subseteq [\![\mathcal{S}_1]\!]$. Then $\preccurlyeq$ is a preorder and the induced equivalence relation is denoted by $\varepsilon_{\preccurlyeq}$. Recall that a poset is called *bounded complete* [14] if any pair of consistent elements has a least upper bound.

Proposition 2 *For a bounded complete poset A, $\langle \mathcal{P}^{\exists\lambda}(A), \preccurlyeq\rangle/\varepsilon_{\preccurlyeq} \cong \mathcal{P}^{\forall}(A)$.*

Proof. If A is bounded complete, then for two finite sets U and L the set $\min(\uparrow U \cap \uparrow L)$ is also finite. Hence, we define $\psi : \mathcal{P}^{\exists\lambda}(A) \to \mathcal{P}^{\forall}(A)$ by $\psi((U, \mathcal{L})) = (U, \{\min(\uparrow U \cap \uparrow L)\} | L \in \mathcal{L})$. Clearly, $[\![\mathcal{S}]\!] = [\![\psi(\mathcal{S})]\!]$ and $\psi(\psi(\mathcal{S})) = \psi(\mathcal{S})$. According to proposition 1, $\psi(\mathcal{S})$ is the only snack in the $\varepsilon_{\preccurlyeq}$-equivalence class of $\mathcal{S}$. Moreover, ψ is monotone because, if $U \sqsubseteq^\sharp V$ and $L \sqsubseteq^\sharp M$, then $\min(\uparrow L \cap \uparrow U) \sqsubseteq^\sharp \min(\uparrow M \cap \uparrow V)$. This finishes the proof of the proposition. □

The following result follows directly from the definitions.

Proposition 3 *Given $\mathcal{S} \in \mathcal{P}^{\emptyset}(A)$, $[\![\mathcal{S}]\!] \neq \emptyset$ iff $\mathcal{S} \in \mathcal{P}^{\exists\lambda}(A)$.* □

Summing up, scones are the maximal class of approximation constructs with multi-set L-part that has well-defined semantics, and snacks are the maximal subclass of scones over on which the semantics and the orderings agree.

4 Universality properties of approximations

Now that we formalized the notion of approximation and found a number of models to represent them, we are about to prove the main technical results of this paper. These results describe most formal models of approximations via their universality properties, or show the absence thereof. As was explained in subsection 2.2, this makes the approximation constructs first class citizens in a query language, provides query language primitives to work with them and suggests a query language syntax.

Due to the nature of the approximation constructs, the characterization theorems and equational theories below are rather involved. For the reader who wants to understand the flavor of the results and then move on to section 5 dealing with query languages for approximations, we included a short subsection below that summarizes the results of this section.

4.1 The flavor of the results and summary

Let us give a quick overview of the universality results. The desired result is 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 for the following reason. Let $x \uparrow y$, where $x, y \in A$. Then $S_{xy} = (\{x\}, \{y\})$ is a sandwich and $\mathcal{S}_{xy} = (\{x\}, \{\{y\}\})$ is a scone. 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 $\eta(\cdot)$. 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

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

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 functions $\eta^\uparrow(x, y) = (\{x\}, \{y\})$ and $\eta^\uparrow(x) = (\{x\}, \{\{y\}\})$. Since $A \uparrow A$ interacts in a certain way with the structure of approximations, we shall seek results 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 $A \uparrow A$ *with respect to the class $\mathcal{C}$ of monotone maps.*

The results of this section are summarized in the following table. For each construct with $u \leq l$ used in the consistency condition (with one exception) we find a free algebra characterization. For constructs with $u \uparrow l$ used in the consistency condition, we show that they do not arise as free algebras generated by the poset itself, but do arise as free constructions generated by $A \uparrow A$ (with respect to a restricted class of map). We use **dna** (does not apply) for constructions based on the $u \leq l$ consistency condition with $A \uparrow A$ as the generating poset. Notice that there are still three **ni** null values – these questions remain open. Nonnull entries give the name of an algebra and refer to the subsection where the result is to be found.

L-part; generator	*type of consistency condition (quantifier–condition)*				
	$\forall \quad u \leq l$	$\forall \quad u \uparrow l$	$\exists \quad u \leq l$	$\exists \quad u \uparrow l$	no condition
one set; A	mix (4.2)	**ne** (4.3)	bi-LNB (4.4)	**ni**	bi-mix (4.5)
one set; $A \uparrow A$	**dna**	mix (4.3)	**dna**	**ni**	**dna**
family of sets; A	snack (4.6)	**ne** (4.7)	**ne** (4.8)	**ne** (4.9)	salad (4.10)
family of sets; $A \uparrow A$	**dna**	**ni**	**dna**	scone (4.9)	**dna**

dna = does not apply; **ne** = non-existent; **ni** = no information (unknown)

For our characterizations, we need two kinds of algebras defined in [32]. A *bisemilattice* $\langle B, +, \cdot \rangle$ is an algebra with two semilattice operations, i.e. idempotent, commutative and associative. It is called *distributive* if both distributive laws hold. A *left normal band* $\langle B, * \rangle$ is an algebra with an idempotent associative operation $*$ such that $x * y * z = x * z * y$.

We shall use four kinds of operations on the approximation constructs. The union-like operations will satisfy the laws of semilattices. An example of such operation is $(U, L) + (V, M) = (\min(U \cup V), \max(L \cup M))$ on mixes or sandwiches. The unary (modal) operations will be used to ignore one of the components of an approximation; for example, we shall use the operation $\Box(U, L) = (U, \emptyset)$ on mixes. We shall also make use of "skewed" union operations that satisfy the left normal band laws. An example of such operation is $(U, L) \oplus (V, M) = (\min(U \cup V), L)$ on elements of $\mathcal{P}^{\exists}(A)$. For approximations $\boldsymbol{\mathcal{P}}^i(A)$, we shall also use pairwise union operations

$$
\begin{tikzcd}
A \arrow[r, hook, "\eta"] \arrow[dr, "f"'] & \langle \mathcal{P}^i(A) \text{ or } \boldsymbol{\mathcal{P}}^i(A), \Omega \rangle \arrow[d, dashed, "\exists! f^+"] \\
& \langle \mathcal{A}, \Omega \rangle
\end{tikzcd}
\qquad
\begin{tikzcd}
A{\uparrow}A \arrow[r, hook, "\eta^{\uparrow}"] \arrow[dr, "f \text{ in } \mathcal{C}"'] & \langle \mathcal{P}^i(A) \text{ or } \boldsymbol{\mathcal{P}}^i(A), \Omega \rangle \arrow[d, dashed, "\exists! f^+"] \\
& \langle \mathcal{A}, \Omega \rangle
\end{tikzcd}
$$

Figure 3: Universality results

that take component-wise unions of the sets in the lower approximation. For details, see section 4.6.

For the rest of the section we use the following notation. To distinguish orderings on algebras and their generating posets, we use $\leq$ for the former and $\precsim$ for the latter. 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 occasionally omit commas separating elements of sets, writing xyz for $\{x, y, z\}$.

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

Define a *mix* algebra [13] $\langle M, +, \Box, e \rangle$ as an algebra with a 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) $x + \Box y \leq x$.
To make $\mathcal{P}^{\forall}(A)$ a mix algebra, interpret the ordering as $\sqsubseteq^{\mathbb{B}}$. 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 ([13]) *$\mathcal{P}^{\forall}(A)$ is the free mix algebra generated by A.* $\Box$

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

First, we present a negative result.

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

Proof. 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^+ cannot 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. $\Box$

However, we can overcome this by using the consistent closure and mix algebras with the same interpretation of operations. Let M be a mix algebra. A monotone map $f : A{\uparrow}A \to M$ is called *sandwich-admissible* if $f(x,y) + f(z,y) \le f(x,y)$ 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 sandwich-admissible maps.*

Proof. Throughout this proof, by admissible we mean sandwich-admissible. We omit an easy verification that $\mathcal{P}^{\forall\wedge}(A)$ is a mix algebra. Now we must show that, given a mix algebra M and an admissible map $f : A{\uparrow}A \to M$, there exists a unique mix homomorphism $f^+ : \mathcal{P}^{\forall\wedge}(A) \to M$ such that the following diagram commutes:

$$\begin{tikzcd}
A{\uparrow}A \arrow[r, hook, "\eta^{\uparrow}"] \arrow[dr, "f"'] & \langle \mathcal{P}^{\forall\wedge}(A), +, \Box, e\rangle \arrow[d, dashed, "\exists! f^+"] \\
 & \langle M, +, \Box, e\rangle
\end{tikzcd}$$

Let us first list a number of useful properties of admissible maps $f : A{\uparrow}A \to M$.

1) Assume $v \precsim u$ and $u{\uparrow}l$. Then $f(u,l) + f(v,l) = f(v,l)$.
2) Assume $p \succsim l$, $v{\uparrow}l$ and $q{\uparrow}p$. Then $f(v,l) + f(q,p) = \Box f(v,v) + f(q,p)$.
3) If $l \precsim m$, then $f(v,l) + f(q,m) = \Box f(v,v) + f(q,m)$.
4) Assume $v \precsim u$. Then $f(v,l) = f(u,l) + \Box f(v,v)$.
5) If $v \succsim u$, then $\Box f(u,u) + \Box f(v,v) = \Box f(v,v)$.
6) Assume $u{\uparrow}l$ and $v{\uparrow}l$. Then $f(v,l) + \Box f(u,u) = f(v,l) + \Box f(u,u) + f(u,l)$.

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

$$\text{(E1)} \qquad \mathcal{S} = \sum_{(i,j)\in\mathcal{I}} \eta^{\uparrow}(u_i, l_j) + \Box \sum_{i=1}^{n} \eta^{\uparrow}(u_i, u_i)$$

From now on we assume that summation over an empty set is the identity for the $+$ operation. It shows that (E1) holds even if one of the components of a sandwich is empty.

Using representation (E1), define f^+ for an admissible $f : A\uparrow A \to M$ as follows:

$$\text{(E2)} \qquad f^+(S) = \sum_{(i,j)\in\mathcal{I}} f(u_i, l_j) + \Box \sum_{i=1}^{n} f(u_i, u_i)$$

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 $\mathcal{S}_1 \sqsubseteq^{\mathbb{I}} \mathcal{S}_2$, that is, $U \sqsubseteq^{\sharp} V$ 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 the claim follows easily from (E1), 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 [25]. 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 $u' \uparrow l'$. The former 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 latter 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 $u \uparrow l$. Then $u \uparrow l'$ for any $l' \in L'$. Therefore, any summand $f(u, l)$ in (E2) for (U, L_{i+1}) is bigger than $f(u, l')$ in (E2) for (U, L_i). Now suppose there is $l' \in L'$ such that $u' \uparrow l'$ but u' is not consistent with l. If l is consistent with some $u \in U$, then $u \uparrow l'$. 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 (E1). Given a sandwich (W, N) and $n \in N$, let w_n be arbitrarily chosen element of W such that $w_n \uparrow n$. Then, given an admissible function f, $f^+(W, N)$ defined by (E2) 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 $m \in M$ and u_m be an element of U under v_m. Then $u_m \uparrow m$. 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 (E2) 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) such that $u_i \in U$, $l_j \in L$ and $u_i \uparrow l_j$. There are three cases: this pair is either present in the representation (E1) of $\mathcal{S}$ or $u_i \gtrsim v_k$ for some $v_k \in V \cap \min(U \cup V)$ or $l_j \lesssim m_k \in M \cap \max(L \cup M)$.

Consider the second case. We have $v_k \uparrow l_j$. Assume $l_j \lesssim p$ and $p \in N$. We know that $p \uparrow q$ 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 $m_k \uparrow q$ 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 $u \gtrsim v$. 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 $u_i \uparrow m_j$. There exists v_k such that $v_k \uparrow m_j$. 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 (E1). Since $f^+(\eta^\uparrow(x,x)) = f(x,x) + \Box f(x,x) = f(x,x)$, we have $f^+ \circ \eta^\uparrow = f$. The theorem is proved. $\Box$

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

An algebra $\langle B, \oplus, * \rangle$ is called a *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) $(a * b) \oplus b = (b * a) \oplus a$.

This definition does not include any notion of order, because the ordering on carriers of bi-LNB algebras can be defined from its operations.

Lemma 1 *In a bi-LNB algebra define $a \le b$ iff $b \oplus a = a * b$. Then $\le$ is a partial order. Moreover, $\oplus$ and $*$ are monotone with respect to $\le$.*

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, $\le$ is reflexive. To prove transitivity, let $a \le b$ and $b \le c$. 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 $a \le c$. Finally, if $a \le b$ and $b \le a$, then $a \oplus b = a$ and $b * a = b$. Hence, $b = b * a = a \oplus b = a$, which finishes the proof that $\le$ is a partial order.

Assume that $a \le b$. To see that $a\oplus c \le b\oplus c$, 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 $a * c \le b * c$, 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. $\square$

From now on, bi-LNB algebras are treated as ordered algebras with the order relation being defined as in lemma 1. The operations $\oplus$ and $*$ on $\mathcal{P}^{\exists}(A)$ as interpreted as follows:

$$(U, L) \oplus (V, M) \;=\; (\min(U \cup V), L) \quad \text{and} \quad (U, L) * (V, M) \;=\; (U, \max(L \cup M)).$$

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

Proof. We leave it to the reader to prove that $\mathcal{P}^{\exists}(A)$ satisfies all equations of the bi-LND algebras under the given interpretation of $\oplus$ and $*$ and that $\mathcal{S}_1 \sqsubseteq^{\mathbb{B}} \mathcal{S}_2$ iff $\mathcal{S}_1 * \mathcal{S}_2 = \mathcal{S}_2 \oplus \mathcal{S}_1$. We must show that for any bi-LNB algebra B and any monotone map $f : A \to B$, there exists a unique homomorphism f^+ such that $f^+ \circ \eta = f$. Observe that if $(U, L) \in \mathcal{P}^{\exists}(A)$, then $U, L \neq \emptyset$. 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 $\sum$ for repeated applications of $\oplus$, and $\bigotimes$ for repeated applications of $*$, we can see that

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

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:

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

In this representation any number of expressions of form $f(u') * f(l')$, where $u' \precsim l'$, can be added after $f(u_1)*f(l_1)$. Since $f(u') \le f(l')$, we have $f(u')*f(l') = f(l')$, 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 $u_i \le l_i$, $i = 1, 2$. 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 U∪$u'\widetilde{U \cup u'}, then$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, $f^+(\mathcal{S}_1) * f^+(\mathcal{S}_2) = \tilde{U} * f(u_1) * \hat{L} * \hat{M} = \tilde{U} * f(u_1) *$ L∪M$\widehat{L \cup M} = f^+(\mathcal{S}_1 * \mathcal{S}_2)$. 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 $f^+(\mathcal{S}_1) \oplus f^+(\mathcal{S}_2) = (\tilde{U} \oplus \tilde{V}) * f(u_1) * \hat{L} =$ U∪V$\widetilde{U \cup V} * f(u_1) * \hat{L} = f^+(\mathcal{S}_1) \oplus f^+(\mathcal{S}_2)$. Thus, f^+ is a homomorphism. This proves Theorem 4. $\square$

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

An algebra $\langle B, \leq, +, \Box, \Diamond \rangle$ 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 $x + \diamond y \geq x$.

We interpret the operations $+, \Box$ and e on $\mathcal{P}^{\emptyset}(A)$ in the same way as we interpreted them for the mix algebras. For the new operation $\diamond$, define $\diamond(U, L) = (\emptyset, L)$.

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

We omit the proof of this theorem, which is very similar, but somewhat simpler, than the proof of theorem 10.

4.6 Universality of $\mathbb{P}^{\forall}(A)$ (snacks)

As we have said before, snacks and mixes are the only two constructs for which universality results are known. For snacks, in the totally unordered case it was first obtained more than 20 years ago, see [30]. Later it was extended to the ordered case in [31]; however, the equational theory used in [31] is slightly different. We now formulate the result and sketch the proof.

A *snack* algebra is a distributive bisemilattice $\langle B, +, \cdot, e\rangle$ with added identity for $+$. That is, $x + e = e + x = x$. Each semilattice operation gives rise to an ordering. We always consider bisemilattices as algebras ordered by the $\cdot$ meet-semilattice operation. That is, $x \leq y$ iff $x \cdot y = x$.

The operations are interpreted as follows:

$$(U, \mathcal{L}) + (V, \mathcal{M}) = (\min(U \cup V), \max^{\sharp}(\mathcal{L} \cup \mathcal{M}))$$

$$(U, \mathcal{L}) \cdot (V, \mathcal{M}) = (\min(U \cup V), \max^{\sharp}\{\min(L \cup M) \mid L \in \mathcal{L}, M \in \mathcal{M}\}),$$

where $\max^{\sharp}$ means family of maximal elements w.r.t. $\sqsubseteq^{\sharp}$. For this interpretation of $\cdot$ on $\mathbb{P}^{\forall}(A)$, the ordering on $\mathbb{P}^{\forall}(A)$ coincides with $\sqsubseteq^{\natural}_{f}$, see [31]. The constant e is interpreted as $(\emptyset, \{\emptyset\})$.

Theorem 6 (see also [30, 31]) *$\mathbb{P}^{\forall}(A)$ is the free snack algebra generated by A.*

Proof sketch. First, $\mathbb{P}^{\forall}(A)$ is a snack algebra [31]. We have to show that for any snack algebra Sn and a monotone map $f : A \to Sn$, there exists a unique snack homomorphism $f^{+} : \mathbb{P}^{\forall}(A) \to Sn$ that extends f.

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

(E3) $$\mathcal{S} = (\prod_{i=1}^{n} \eta(u_i))e + \sum_{i=1}^{k} \prod_{j=1}^{k_i} \eta(l^i_j)$$

Then, if a monotone $f : A \to Sn$ is given, define $f^{+} : \mathbb{P}^{\forall}(A) \to Sn$ by $f^{+}(\mathcal{S}) = (\prod_{i=1}^{n} f(u_i))e + \sum_{i=1}^{k} \prod_{j=1}^{k_i} f(l^i_j)$. Clearly, $f^{+}(\emptyset, \emptyset) = e$ and $f^{+}(\eta(x)) = f(x) \cdot e + f(x) = f(x)$. It is fairly routine to show that f^{+} is the unique homomorphic extension of f. $\Box$

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

We have seen that the union-like operation $+$, that takes the component-wise union of two approximation constructs, is present in all characterizations obtained so far. One can also see that all ten approximation constructs are closed under this operation. Thus, it is natural to require that $+$ be among the operations associated with approximations. However, no such set operations can be found for $\mathcal{P}^{\forall\wedge}(A)$.

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

Proof. 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 $f : A \to \mathcal{P}^{\forall\wedge}(A')$ 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 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

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

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

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

As with the case of $\mathcal{P}^{\forall\wedge}(A)$, we can show that no set of operations from which $+$ is derivable supplies $\mathcal{P}^{\exists}(A)$ with the structure of a free algebra generated by A.

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

Proof. 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, $\langle \mathcal{P}^{\exists}(B), \Omega_+\rangle$ is the free Ω_+ algebra generated by B. Then we would have a monotone $+$-homomorphism f^+ :

$\mathcal{P}^{\exists}(A) \to \mathcal{P}^{\exists}(A')$ 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 $(x,xy) \le (x,x)$, 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 $(y,xy) \le (y,y)$, $f^+((y,xy))$ 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 $\mathcal{P}^{\exists}(A)$ and $\mathcal{P}^{\exists}(A')$ and hence $\mathcal{P}^{\exists}(A)$ is not a free Ω_+-algebra generated by A. $\Box$

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

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.

Similarly to the case of $\mathcal{P}^{\exists}(A)$, one can use the operations of the scone algebras to define the order relation on them. The following is immediate from the equational theory of the scone algebras.

Lemma 2 *In a scone algebra, $x \cdot y = x * y + y * x$ is a semilattice operation.* $\Box$

The order on scone algebras will be defined by $x \le y$ iff $x \cdot y = x$.

The operation $+$ and the constant e are interpreted as for snacks. The operation $*$ is interpreted as

$$(U,\mathcal{L}) * (V,\mathcal{M}) = (U, \max^{\sharp}\{\min(L \cup M) \mid L \in \mathcal{L}, M \in \mathcal{M}\}).$$

Now it can be seen that for $\cdot$ defined in lemma 2, $(U,\mathcal{L}) \cdot (V,\mathcal{M})$ coincides with the meet operation $\cdot$ given for snacks in subsection 4.6. In particular, for this interpretation of the operations, the interpretation of the ordering is $\sqsubseteq_f^{\natural}$.

We shall give two different characterization of $\mathcal{P}^{\exists\curlywedge}(A)$ as scone algebras, one generated by A and the other by $A{\uparrow}A$. For this, we need two different definitions of admissibility.

Let $\langle Sc, +, *, e\rangle$ be a scone algebra. A monotone map $f : A{\uparrow}A \to Sc$ 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 $x \uparrow y_1$ and $x \uparrow y_2$ such that $x, y_i \le z_i, i = 1,2$, 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 $A\uparrow A$ 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.* $\square$

Proof of part 1. 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, $\mathcal{P}^{\exists\wedge}(A)$ is a scone algebra.

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 $f^+ : \mathcal{P}^{\exists\wedge}(A) \to Sc$ such that $f^+ \circ \eta^\uparrow = f$. We need some some facts about the scone algebras. In what follows, $f : A\uparrow A \to Sc$ is an admissible map. The first equation for admissibility can be rewritten as $f(u,l) * f(v,m) = f(u,l) * f(w,m) = f(u,m) * f(v,l)$. The easy proofs of 1)–8) below are omitted.

1) $+$ is monotone with respect to the ordering given by $\cdot$.
2) $\cdot$ distributes over $+$.
3) If $a \le b$, then $a * e \le b * e$.
4) $f(x,y) + f(z,y) \le f(x,y)$.
5) If $a \precsim b$, then $f(a,a) * e + f(b,b) * e = f(a,a) * e$.
6) If $a \precsim b$ and $b\uparrow x$, then $f(x,a) * f(b,b) = f(x,a)$.
7) For any $a\uparrow b$, $f(a,b) * f(b,a) \le f(a,b)$.
8) If $a \precsim b$, then $f(b,b) * f(a,a) = f(b,a)$.

Let $\mathcal{S} = (U,\mathcal{L})$ be a scone over A. Since $\uparrow U \cap \uparrow L_j \ne \emptyset$ for all $L_j \in \mathcal{L}$, there exists a pair $(u_i, l^j_{k_i})$ for every j such that $u_i \uparrow l^j_{k_i}$. Let $i(j)$ and $k(j)$ be some indices such that $u_{i(j)} \uparrow l^j_{k(j)}$. Then $\mathcal{S}$ can be represented as

$$\text{(E4)} \qquad \mathcal{S} = \sum_{u\in U} \eta^\uparrow(u,u) * e + \sum_{L_j\in\mathcal{L}} (\eta^\uparrow(u_{i(j)}, l^j_{k(j)}) * \bigotimes_{l\in L_j} \eta^\uparrow(l,l))$$

Recall that we use $\otimes$ for repeated applications of $*$, and 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, (E4) does not depend on how the pairs $(i(j), k(j))$ are chosen.

Using (E4), define

$$\text{(E5)} \qquad f^+(\mathcal{S}) = \sum_{u\in U} f(u,u) * e + \sum_{L_j\in\mathcal{L}} (f(u_{i(j)}, l^j_{k(j)}) * \bigotimes_{l\in L_j} f(l,l))$$

Our first goal is to verify that f^+ is well-defined, that is, it does not depend on how the 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 $q+p \leq r+p$. First, we prove $p \leq r$. 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, $p \leq r$. Similarly, we can show that $p \leq q$.

To prove $q+p \leq r+p$, calculate, using 2), $(q+p)(r+p) = rq+rp+qp+p$. Since $p \leq r$ and $p \leq q$, we obtain $(q+p)(r+p) = rq+p$. By monotonicity of $+$, we have $rq+p \leq q+p$. Assume we prove $q+qr = qr$; then $q+p = q+q+p \leq q+qr+p = qr+p$. Hence, it remains to show $q+q*r+r*q = q*r+r*q$. Calculate the left hand side: $q+q*r+r*q = f(u,l)*f(l',l')+f(u,l)*f(l',l')*f(u',l')*f(l,l)+f(u',l')*f(l,l)*f(u,l)*f(l',l') =$ (by admissibility of f) $= f(u,l)*f(l',l')+f(u,l)*f(l',l')+f(u',l')*f(l,l) = q*r+r*q$ by idempotency of $+$. 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, (E5) remains true. This makes it easier to prove that f^+ is a homomorphism.

First observe that if $u \in U$ and $v \succsim u$, then $\tilde{U}*e = \mathrm{U} \cup v\widetilde{U \cup} v{*}e$ (we use the 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 $l' \succsim l \in L$, the corresponding summand of (E5) 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 (E5). Assume $u \in U$, $m \in M$ and $l \in L$ are such that $m \leq l$ and $u \uparrow l$ (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 the representation (E5)). 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 $d \succsim c$ 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 (E5).

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 (E5) 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 $u\uparrow l$, $v\uparrow m$, $v \in V$, $u \in U$, $l \in L \in \mathcal{L}$ 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,

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

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 (E4) and well-definedness of (E5). Finally, $f^+(\eta^\uparrow(x,y)) = f(x,x)*e+f(x,y)*f(y,y) = f(x,y)*e+f(x,y) = f(x,y)$. This shows $f^+\circ\eta^\uparrow = f$. 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 f^+ such that $f^+\circ\eta = f$. Define $\varphi_f : A\!\uparrow\! A \to Sc$ 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-admissibility 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 $z \succsim x,y$.

Let $\Delta : A \to A\!\uparrow\! A$ be given by $\Delta(a) = (a,a)$. Note that $\varphi_f\circ\Delta = f$: $\varphi_f((x,x)) = (f(x)*e+f(x))*f(x) = f(x)*e+f(x) = f(x)$.

Claim. φ_f is admissible (according to definition before this theorem).

Before we prove this, let us show how the theorem follows from the claim. Consider this diagram:

$$
\begin{tikzcd}
A \arrow[r, "\Delta"] & A{\uparrow}A \arrow[r, "\eta^{\uparrow}"] \arrow[dr, "\varphi_f"'] & \mathcal{P}^{\exists\curlywedge}(A) \arrow[d, dashed, "\exists! f^{+}"] \\
& & Sc
\end{tikzcd}
$$

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

Proof of the claim. 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 $a \le b$. Since $f(x) \le f(z_1)$, 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 $b \ge c$ 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 the scone-admissibility of f. This proves the claim and part 2.

Proof of part 3. 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. Consider the scone algebra $Sc_1 = \langle B, +, *, e\rangle$ whose 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 (in fact, 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 $f(y) = p_4$. Suppose that f can be extended to a homomorphism $f^+ : \mathcal{P}^{\exists\curlywedge}(A) \to Sc$. Then

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

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. This proves part 3 and theorem 9. $\Box$

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

A *salad algebra* $\langle Sd, +, \cdot, \Box, \Diamond\rangle$ is an algebra with two semilattice operations $+$ and $\cdot$ and two unary operations $\Box$ 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) $\Box\Box x = \Box x$.

The binary operations $+$ and $\cdot$ are interpreted as for snacks, and the unary operations $\Box$ and $\Diamond$ are interpreted as for $\mathcal{P}^{\emptyset}(A)$. The order relation is defined as for the snacks. That is, $x \leq y$ iff $x \cdot y = x$.

Define $\Box Sd = \{\Box x \mid x \in Sd\}$ and $\Diamond Sd = \{\Diamond x \mid x \in Sd\}$. Some useful properties of salad algebras are summarized in the following proposition.

Proposition 4 *Given a salad algebra Sd, the distributivity law $x + yz = (x+y)(x+z)$ holds. Consequently, $+$, $\Box$ and $\Diamond$ are monotone. In addition, the following holds:*
(i) $\Box x \leq x \leq \Diamond x$.
(ii) *$\Diamond Sd$ is a distributive lattice.*
(iii) *$+$ and $\cdot$ coincide on $\Box Sd$.*
(iiii) $\Box\Diamond x = \Diamond\Box y$.

Proof. Using 2) and distributivity law 1) calculate $(x+y)(x+z) = (\Box x + \Box y + \Diamond x + \Diamond y)(\Box x + \Box z + \Diamond x + \Diamond z) =$ (by 1) and 6)) $= \Box x + \Box y + \Box z + \Diamond x + \Diamond x \cdot \Diamond y + \Diamond x \cdot \Diamond z + \Diamond y \cdot \Diamond z =$ (by 7)) $= \Box x + \Box y + \Box z + \Diamond x + \Diamond y \cdot \Diamond z$. Similarly, $x + yz = \Box x + \Diamond x + (\Box y + \Diamond y)(\Box z + \Diamond z) = \Box x + \Diamond x + \Box y + \Box z + \Diamond y \cdot \Diamond z$. Hence, $(x+y)(x+z) = x + yz$. Now monotonicity of $+$ follows from the distributivity laws. That $\Box$ and $\Diamond$ are monotone, follows from 4) and 6). To prove (i), calculate $x \cdot \Box x = (\Box x + \Diamond x)\Box x = \Box x + \Diamond x \cdot \Box x = \Box x + \Box x = \Box x$. Moreover, $x \cdot \Diamond x = (\Box x + \Diamond x)\Diamond x = \Box x \cdot \Diamond x + \Diamond x = \Box x + \Diamond x = x$.

(ii) and (iii) follow immediately from the definitions.

(iiii) By 7), $\Box x \leq \Diamond\Box y$; hence $\Diamond\Box x \leq \Diamond\Box y$ and by symmetry $\Diamond\Box x = \Diamond\Box y$. Similarly, $\Box\Diamond x = \Box\Diamond y$. Define $e_\Diamond = \Diamond\Box x$ and $e_\Box = \Box\Diamond x$. The equations above show that $e_\Diamond$ and $e_\Box$ are well-defined. Now calculate $e_\Diamond + x = \Diamond\Box x + x = \Diamond\Box x + \Diamond x + x = \Diamond(\Box x + x) + x = \Diamond x + x = x$.

Similarly, $e_\Box + x = \Box\Diamond x + x = \Box\Diamond x + \Box x + x = \Box(\Diamond x + x) + x = \Box x + x = x$. Thus, both $e_\Diamond$ and $e_\Box$ are identities for $+$. Therefore, $e_\Diamond = e_\Diamond + e_\Box = e_\Box$. $\Box$

This proposition tells us that we can give the following equivalent definition of a salad algebra: A salad algebra is a distributive bisemilattice $\langle Sd, +, \cdot \rangle$ on which a projection $\Box$ and a closure $\Diamond$ are defined such that $\Box Sd$ is a semilattice, $\Diamond Sd$ is a lattice, $x = \Box x + \Diamond x$ and $\forall x \in \Box Sd\, \forall y \in \Diamond Sd$: $x \le y$.

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

Proof. We 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

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

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 we 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 we show that $\mathcal{P}^\emptyset(A)$ is free. That is, 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 $f^+ \circ \eta = f$.

Given a salad $\mathcal{S} = (U, \mathcal{L})$,

$$\mathcal{S} = \Box \sum_{u \in U} \eta(u) + \Diamond \sum_{L \in \mathcal{L}} \prod_{l \in L} \eta(l) \tag{E6}$$

To see that this also works for empty components, observe that $\Box e = \Diamond e = e$. Now, given monotone $f : A \to Sd$, define

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

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 proposition 4) 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

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

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 $x' \succsim x \in X$. 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,

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

Thus, f^+ is a homomorphism. Its uniqueness follows from (E6). Theorem is proved. $\Box$

Summing up, there are four kinds of operations naturally associated with the approximation constructs: union operations (like $+$), pairwise union operations (like $\cdot$), skewed versions of the above (like $\oplus$ and $*$) and modal operations (like $\Box$ and $\Diamond$).

4.11 Relationship between the approximations

In this subsection we study the relationship between the four previously known approximations: mixes, sandwiches, scones, and snacks. Others may be included as well, but this will make diagrams incomprehensible, so we limit our attention to the examples that motivated this study. We also show that we can view all four as instances of the most general construction: salads $\mathcal{P}^\emptyset(\cdot)$. We will explain that by their "complexity" the approximation constructs "decrease" as

$$\text{Salads} \;\rightarrow\; \text{Scones} \;\rightarrow\; \text{Snacks} \;\rightarrow\; \text{Sandwiches} \;\rightarrow\; \text{Mixes}$$

and algebras as

$$\text{Salads} \;\rightarrow\; \text{Scones} \;\rightarrow\; \text{Snacks} \;\rightarrow\; \text{Mixes}$$

Relationship between algebras. The general technique we use is the following. Given an algebra $\langle \mathcal{A}, \Omega \rangle$, let Ω' be a subset of Ω and Ω'' a set of derived operations. Let $\Theta = (\Omega - \Omega') \cup \Omega''$. Then $\mathcal{A}$ can be considered as a Θ-algebra which is called Θ-*reduct of* $\langle \mathcal{A}, \Omega \rangle$. We denote a map that takes an Ω-algebra $\langle \mathcal{A}, \Omega \rangle$ and returns the Θ-algebra $\langle \mathcal{A}, \Theta \rangle$ by $\varphi^{\Omega \to \Theta}$.

For reductions for the algebras from the previous section, we use the same superscripts as for the approximation constructs themselves, except that we use index f (family) for $\mathcal{P}^{i}$'s. For example, a snack reduct of a scone will be denoted by $\varphi^{\exists\wedge \to \forall_f}$.

Definition. a) *Given a salad algebra* $Sd = \langle \mathcal{A}, +, \cdot, \Box, \Diamond \rangle$*, define its reducts as follows:*

Scone reduct $\varphi^{\emptyset \to \exists\wedge}(Sd) = \langle \mathcal{A}, +, *, e \rangle$ *where* $x * y = x \cdot \Diamond y$ *and* $e = \Diamond \Box x$.

Snack reduct $\varphi^{\emptyset \to \forall_f}(Sd) = \langle \mathcal{A}, +, \cdot, e \rangle$ *where* $e = \Diamond \Box x$.

Mix reduct $\varphi^{\emptyset \to \forall}(Sd) = \langle \mathcal{A}, +, \Box, e \rangle$ *where* $e = \Diamond \Box x$.

b) *Given a scone algebra* $Sc = \langle \mathcal{A}, +, *, e \rangle$*, define its reducts as follows:*

Snack reduct $\varphi^{\exists\wedge \to \forall_f}(Sc) = \langle \mathcal{A}, +, \cdot, e \rangle$ *where* $x \cdot y = x * y + y * x$.

Mix reduct $\varphi^{\exists\wedge \to \forall}(Sc) = \langle \mathcal{A}, +, \Box, e \rangle$ *where* $\Box x = x * e$.

c) *Given a snack algebra* $Sn = \langle \mathcal{A}, +, \cdot, e \rangle$*, define its* mix reduct $\varphi^{\forall_f \to \forall}(Sn)$ *as* $\langle \mathcal{A}, +, \Box, e \rangle$ *where* $\Box x = x \cdot e$.

Our first goal is to show that the concepts above are well-defined, i.e. that a mix reduct is a mix algebra, a scone reduct is a scone algebra etc. We then proceed to prove path independence. That is, it does not matter if we perform reduction from one algebra to another directly or via a number of steps.

Proposition 5 *The reducts above are well-defined.*

Proof sketch. We give the proof that $\varphi^{\emptyset \to \exists\wedge}(Sd)$ is a scone algebra; others can be proved in a similar fashion. That e is the identity for $+$ was already proved. Distributivity of $*$ over $+$ is obvious. We must show the other distributivity law: $a + x * y = (a + x) * (a + y)$. To prove this, calculate $a + xa = a + (\Box x + \Diamond x)(\Box a + \Diamond a) = a + \Box x \cdot \Box a + \Box x + \Box a + \Diamond x \cdot \Diamond a = a + \Box x + \Diamond x \cdot \Diamond a = a + (\Box x + \Diamond x) \Diamond a = a + a \cdot \Diamond a$. Now, $a + x * y = a + x \cdot \Diamond y = (a + x)(a + \Diamond y) = a + xa + a \cdot \Diamond y + x \cdot \Diamond y = a + x \cdot \Diamond a + a \cdot \Diamond y + x \cdot \Diamond y = (a + x)(\Diamond a + \Diamond y) = (a + x) * (a + y)$. This proves distributivity. That $*$ is a left normal band operation is obvious. We have $e * x = \Diamond \Box x \cdot \Diamond x = \Diamond(\Box x \cdot x) = \Diamond \Box x = e$. Finally, $x + x * y = x + (\Box x + \Diamond x) \cdot \Diamond y = x + \Box x + \Diamond x \cdot \Diamond y = x + \Box x + \Diamond x + \Diamond x \cdot \Diamond y = x + \Box x + \Diamond x = x$. Therefore, $\varphi^{\emptyset \to \exists\wedge}(Sd)$ is a scone algebra. $\Box$

The path independence result can be formalized as follows.

Theorem 11 *The following diagram commutes (where the arrow from* Sd *to* Sn *is* $\varphi^{\emptyset \to \forall_f}$ *and the arrow from* Sc *to* Mix *is* $\varphi^{\exists\wedge \to \forall}$*):*

$$
\begin{tikzcd}
Sd \arrow[r, "\varphi^{\emptyset\to\exists\wedge}"] \arrow[d, "\varphi^{\emptyset\to\forall}"'] \arrow[dr] & Sc \arrow[d, "\varphi^{\exists\wedge\to\forall_f}"] \arrow[dl] \\
Mix & Sn \arrow[l, "\varphi^{\forall_f\to\forall}"]
\end{tikzcd}
$$

The proof of this theorem is given by straightforward calculations, and is omitted here.

Embeddings. The reductions above correspond to the embeddings of the approximation constructs. Assume that a poset A is given and $\mathcal{P}'$ and $\mathcal{P}''$ are two approximation constructions such that $\mathcal{P}'$ is "higher" than $\mathcal{P}''$ in the hierarchy shown in the beginning of the section. Then there is a reduction φ that takes $\mathcal{P}'(A)$ and makes it an algebra in the signature corresponding to $\mathcal{P}''$. Depending on the generating poset for $\mathcal{P}''(A)$, the subalgebra of $\varphi(\mathcal{P}'(A))$ generated by either $\eta(A)$ or $\eta^{\uparrow}(A)$ is $\mathcal{P}''(A)$. Moreover, this construction is "path independent" in the sense of theorem 11. To formalize it, we use the notation

$$
\mathcal{P}'(A) \xrightarrow{[\eta(A)]\circ\varphi} \mathcal{P}''(A) \quad \text{or} \quad \mathcal{P}'(A) \xrightarrow{[\eta^{\uparrow}(A)]\circ\varphi} \mathcal{P}''(A)
$$

The meaning of these arrows is: Take $\mathcal{P}'(A)$ and consider it as an algebra corresponding to $\mathcal{P}''$ (by means of φ). Then its subalgebra generated by $\eta(A)$ (or $\eta^{\uparrow}(A)$) is $\mathcal{P}''(A)$.

Theorem 12 *In the following diagram all arrows are well-defined and the diagram commutes:*

$$
\begin{tikzcd}
\boldsymbol{\mathcal{P}}^{\emptyset}(A) \arrow[r, "{[\eta^{\uparrow}(A)]\circ\varphi^{\emptyset\to\exists\wedge}}"] \arrow[dd, "{[\eta(A)]\circ\varphi^{\emptyset\to\forall}}"'] \arrow[ddr] \arrow[drr] & \boldsymbol{\mathcal{P}}^{\exists\wedge}(A) \arrow[ddl] \arrow[dd] \arrow[dr, "{[\eta(A)]\circ\varphi^{\exists\wedge\to\forall_f}}"] & \\
 & & \boldsymbol{\mathcal{P}}^{\forall}(A) \arrow[dll] \arrow[dl, "{[\eta^{\uparrow}(A)]\circ\varphi^{\forall_f\to\forall}}"] \\
\mathcal{P}^{\forall}(A) & \mathcal{P}^{\forall\wedge}(A) \arrow[l, "{[\eta(A)]}"] &
\end{tikzcd}
$$

The arrows not shown on the diagram are:

$$[\eta(A)] \circ \varphi^{\emptyset \to \forall_f} : \mathcal{P}^{\emptyset}(A) \to \mathcal{P}^{\forall}(A) \qquad [\eta^{\uparrow}(A)] \circ \varphi^{\emptyset \to \forall} : \mathcal{P}^{\emptyset}(A) \to \mathcal{P}^{\forall\wedge}(A)$$

$$[\eta(A)] \circ \varphi^{\exists\wedge \to \forall} : \mathcal{P}^{\exists\wedge}(A) \to \mathcal{P}^{\forall}(A) \qquad [\eta^{\uparrow}(A)] \circ \varphi^{\exists\wedge \to \forall} : \mathcal{P}^{\exists\wedge}(A) \to \mathcal{P}^{\forall\wedge}(A)$$

$$[\eta(A)] \circ \varphi^{\forall_f \to \forall} : \mathcal{P}^{\forall\wedge}(A) \to \mathcal{P}^{\forall}(A)$$

Proof. Full proof requires a lot of calculations, so we only sketch it here. First observe that all definitions of new operations for reductions agree with their interpretation. For example, given two scones $(U, \mathcal{L})$ and $(V, \mathcal{M})$ in $\mathcal{P}^{\exists\wedge}(A)$, the value of $(U, \mathcal{L}) \cdot (V, \mathcal{M})$ in $\varphi^{\exists\wedge \to \forall_f}(\mathcal{P}^{\exists\wedge}(A))$ is $(U, \mathcal{L}) * (V, \mathcal{M}) + (V, \mathcal{M}) * (U, \mathcal{L}) = (\min(U \cup V), \max^{\sharp}\{L \cup M | L \in \mathcal{L}, M \in \mathcal{M}\})$, which is indeed the infimum operation in $\mathcal{P}^{\forall}(A)$. The verification that other reductions agree with the operations on approximations is also straightforward. Now representations of sandwiches (E1), snacks (E3), scones (E4) and mixes as

$$(U, L) = \Box \sum_{u \in U} \eta(u) + \sum_{l \in L} \eta(l) \tag{E7}$$

tell us that all arrows are well-defined. Commutativity follows in a straightforward way from the representations (E1), (E3), (E4), (E7) and theorem 11. □

5 Programming with approximations

In this section we consider programming with approximations. As we explained before, we want to make the approximation constructs first class citizens in a query language. In particular, we want to view them as *polymorphic type constructor*. That is, for every object type t there is a new type t *mix* such that $[\![t \textit{ mix}]\!] = \mathcal{P}^{\forall}([\![t]\!])$ and a new type t *scone* such that $[\![t \textit{ scone}]\!] = \mathcal{P}^{\exists\wedge}([\![t]\!])$ and so on.

We turn the available universality properties and operations naturally associated with the approximation constructs (see section 4) into programming syntax. We then show that languages thus obtained have a number of drawbacks. In an attempt to overcome their 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 $or{-}\mathcal{NRA}$ of [25] to show how a number of typical problems can be solved.

Encoding approximations in the type system with or-sets gives us more than a purely theoretical result. There exists a system called OR-SML [15], which is a set of libraries on top of Standard ML that implement the types of complex objects and or-sets and some features of a database programming language. We can use the encoding and then program some basic algorithms for querying independent databases in a working systems. Preliminary results of some experiments in this direction can be found in [15].

In this section we shall make use of the *nested relational algebra*, $\mathcal{NRA}$, introduced in subsection 2.2. Recall that $\mathcal{NRA}$'s basic operators are the equality test, conditional *if-then-else*, record formation and projection, set union, cartesian product, singleton formation and the operation *ext* that extends a function from elements of a set to the whole set, cf. [8, 7]. Instead of *ext*, one can use *map*(f) that maps f over all elements of a set, together with μ that flattens a set of sets (that is, takes union of elements), see [8].

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

Figure 4: Structural recursion on mixes (left) and sandwiches (right)

5.1 Using universality properties

We consider only mixes and sandwiches for illustration. Since mixes possess a universality property, we can define structural recursion on them. Similarly, 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.

Structural recursion on mixes and sandwiches 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. As in the case of sets, one might ask if, by setting these parameters in such a way that they do not obey the laws of the equational theory, one may write ill-defined programs. This is indeed the case.

Proposition 6 *It is undecidable whether the structural recursion on mixes or sandwiches is well-defined for a given choice of e, u and h.*

Proof. 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. $\Box$

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. Assume that comparable elements have not been deleted from a pair of sets that represents a mix. That is, a mix (U,L) is represented by a pair (U_1,L_1) such that $U = \min U_1$ and $L = \max L_1$. Note that such a pair (U_1,L_1) is not unique for (U,L). Thus, one would expect that whenever a function f can be applied to (U_1,L_1), it is the case that $f(U_1,L_1)$ yields a representation of $f(U,L)$. But this is not always the case.

To explain why, we use a simpler case of nested relations. As we have just seen, structural recursion on sets can be simulated with mixes of form $(U,\emptyset)$, and thus $\mathcal{NRA}$ can be considered as a sublanguage of the language induced by the construct mix_ext in the same way as $\mathcal{NRA}$ is induced by ext.

Recall that sets are ordered by $\sqsubseteq^\flat$, see subsection 2.1. The way to force sets into antichains is to keep their maximal elements. Indeed, $X \sqsubseteq^\flat Y$ iff $\max X \sqsubseteq^\flat \max Y$, and the semantics of X and $\max X$ coincide. Now assume X_1 and X_2 of type $\{t\}$ are such that $\max X_1 = \max X_2 = X$.

Let f be of type $s \to t$. Since *map* is a part of $\mathcal{NRA}$, it would be desirable if $map(f)(X_1)$ and $map(f)(X_2)$ yield sets Y_1 and Y_2 that represent $Y = map(f)(X)$ in the sense that $\max Y_1 = \max Y_2 = Y$. However, this happens if and only if f is monotone with respect to the order on objects [22]. Thus, monotonicity is needed for well-definedness in the case of antichain semantics.

Theorem 13 *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.*

Proof. Assume that monotonicity is decidable. Given two $\mathcal{NRA}$ functions $f, g : \{s\} \to t$, define a new function $\phi : \{s\} \to \{bool\}$ as $\phi(x) := if\ x = \emptyset\ then\ \{true\}\ else\ if f(x) = g(x)\ then\ \{true\}\ else\ \{false\}$. Then f and g coincide iff $f(\emptyset) = g(\emptyset)$ and ϕ is monotone. Thus having a test for monotonicity would give us equality test for functions of type $\{s\} \to t$. But such functions include all functions definable in the relational algebra, and it is known (cf. [1]) that equality of those is undecidable. $\square$

We can observe the same phenomenon for mixes and other approximations, based on their reduction to nested relations by means of "forgetting" of one of the components. Therefore, turning universality properties into syntax, we encounter a number of problems. First, most operations used in the universality properties for approximations are not as intuitive as union, intersection and so on. Second, all approximations have different equational characterizations, and therefore there are several forms of structural recursion and as many sets of the *ext* primitives. If a language contains all of them, it is going to be too complicated to comprehend. Finally, verification of preconditions is big problem that can not be taken care of by the compiler as the preconditions are undecidable – even for the *ext* operations when the ordered model is used. Therefore, we need a unifying framework for programming with approximations.

5.2 Using or-sets

Or-sets are sets of disjunctive possibilities [17, 25]: an or-set $\langle 1, 2, 3\rangle$ denotes an integer which is 1, or 2 or 3. A language $or\!-\!\mathcal{NRA}$ was proposed in [25]. 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 by picking one element from each or-set in the input. For example, $\alpha(\{\langle 1,2\rangle, \langle 2,3\rangle\}) = \langle\{1,2\},\{1,3\},\{2\},\{2,3\}\rangle$. For technical convenience, we also include operations that convert sets into or-sets and vice versa.

Recall that or-sets are ordered by $\sqsubseteq^\sharp$, see subsection 2.1 and [25]. Thus, we can define an order relation for every object type t, provided such a relation $\leq_b$ is given for every base type b:

- RECORD: $[l_1 = x_1, \ldots, l_n = x_n] \leq_{[l_1:t_1,\ldots,l_n:t_n]} [l_1 = y_1, \ldots, l_n = y_n]$ iff $x_1 \leq_{t_1} y_1$, $\ldots$, $x_n \leq_{t_n} y_n$.
- SET: $X \leq_{\{t\}} Y$ iff $X \leq_t^\flat Y$. That is, $\forall x \in X \exists y \in Y : x \leq_t y$.

- OR-SET: $X \leq_{\langle t \rangle} Y$ iff $X \leq_t^\sharp Y$. That is, $\forall y \in Y \exists x \in X : \; x \leq_t y$.

Recall that the approximation constructs are ordered by $\sqsubseteq^{\mathbb{B}}$ or $\sqsubseteq^{\mathbb{B}}_f$, and these orderings can be compactly represented as $\sqsubseteq^\sharp \times \sqsubseteq^\flat$ and $\sqsubseteq^\sharp \times (\sqsubseteq^\sharp)^\flat$. This suggests the following encoding of the approximation constructs with sets and or-sets:

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

It can be immediately seen from this encoding that the orderings on the encodings of objects of types $t\ mix, t\ snack$ etc are precisely the orderings associated with those approximation constructs. Moreover, there is a close semantic connection between or-sets and approximations that further justifies this connection. This connection makes use of two semantic functions for objects with or-sets (cf. [25]) and we omit it here and refer the reader to [22] for technical details.

To show that this encoding is useful for programming with approximations, denote by $\mathcal{L}_{mix}, \mathcal{L}_{sand}, \ldots$ the language obtained from the restricted form of structural recursion (that is, *ext*) for the mixes, sandwiches etc (that is, for the constructs for which a universality property was established).

Theorem 14 *Assume that each base type b comes equipped with an order relation $\leq_b$ and a test for consistency $\uparrow_b$. Then, using the encoding of approximation constructs with sets and or-sets, the following can be expressed in or$-\mathcal{NRA}$.*

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

Proof sketch. To prove 1 and 2, note that $\sqsubseteq^\sharp$ and $\sqsubseteq^\flat$ are first-order definable and the Buneman orderings are compositions of those. Thus, they are definable in *or*$-\mathcal{NRA}$. It is an easy exercise to see that all operations on approximations that arise from the universality properties are definable. 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*$-\mathcal{NRA}$.

For 3, we consider mixes as an illustration. By f_a we shall denote the antichain analog of a function f, that is, f followed by converting of its output into an antichain-based object. Denote the first and second projections by π_1 and π_2 . For all set operations, there are operations with prefix *or* that act similarly on or-sets. For $f : t \to s\ mix$, where $s\ mix$ is now abbreviation for $\langle s \rangle \times \{s\}$, we have

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

Mix singleton is defined as $\eta\_mix(x) = (or\_\eta, \eta)$; the type of η_mix is $s \to s\ mix$. The proof for other constructions is similar. The functions converting sets into or-sets and vice versa are needed for the multi-element lower approximations. In fact, they are needed to define the converse to α_a. $\square$

While the problem that monotonicity of expressions is undecidable remains for $or-\mathcal{NRA}$, we believe that this language is more suitable for programming with approximations than the collection of languages $\mathcal{L}_\star$. First, its type system is much simpler, and so are the primitives. It is still possible to write ill-defined programs, but using the primities of $or-\mathcal{NRA}$ this appears to be less likely than with primitives such as $mix\_ext(f)$. Second, the number of primitives of $or-\mathcal{NRA}$ is small, and we do not need all primitives $ext\_\star$ as they can be encoded. Again, this makes programming easier. Finally, each expression of $or-\mathcal{NRA}$ *is* well defined. The problem of non-well-definedness does not go away completely: we can have an $or-\mathcal{NRA}$ expression into which an ill-defined program in one of the languages $\mathcal{L}_\star$ is translated. However, this problem no longer concerns the main programming primitives of the language.

Example: removing 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. As an additional benefit of encoding approximations with other datatypes, we demonstrate that we can handle *data anomalies.*

Assume that a query is asked, and it returns a sandwich approximation for another query. However, this answer fails to satisfy the consistency condition of a sandwich. For instance, in the TA example we may get two relations:

Employees:

Name	Salary	Room
John	15K	$\perp$
Mary	12K	$\perp$

CS1:

Name	Salary	Room
John	$\perp$	76
Michael	$\perp$	320

They fail to satisfy the consistency condition of a sandwich because Michael is not an employee. Hence, as the first step, we eliminate this anomaly to force these relations into a sandwich. In what follows, we use functions such as *select*, *cartprod*, Boolean connectives in conditions and so on. We also use one level of λ-abstraction. As follows from [8], all of these are definable in $\mathcal{NRA}$.

To remove anomalies, we only leave those elements in CS1 that are consistent with some element of Employee. First, define the function that selects the subset of element of X compatible with x:

$$compatible \;\equiv\; \lambda(x, X).select(\lambda z.x\uparrow z)(X)$$

Then the function *remove_anomaly* keeps elements of the lower approximation that are compatible with the upper:

$$remove\_anomaly \;\equiv\; \lambda(U, L).(U, select(\lambda x.\neg eq(\emptyset, compatible(x, U)))(L))$$

The idea of the *promotion* operation of [5] was illustrated in the Introduction by extracting a mix from a sandwich. For each element of the lower approximation, as much information as possible is derived about it by using the upper approximation. To do this, we need functions performing order-theoretic join and meet (least upper and greatest lower bounds). We assume that such operations are given for base types. Then they can be derived for all types [6, 22].

Assume that we have them in the form of functions $join, meet : t \times t \to \langle t \rangle$. The result of $join(x, y)$ is $\langle x \vee y \rangle$ if $x \vee y$ is defined and $\langle\rangle$ otherwise, and similarly for $meet(x, y)$. Define

$$big\_meet \;\equiv\; id^{+}[\langle\rangle, \lambda(x, y).or\_ext(\lambda z.meet(\pi_1(z), \pi_2(z)))(cartprod(x, y))]$$

Here $id^{+}[e, u]$ is structural recursion on or-sets with parameters e, u and id. This function calculates the meet of all elements in an or-set.

To define the promotion operation, for each l in the lower approximation L, find the set U_l of all elements in U consistent with l, and calculate $\bigwedge(l \vee u | u \in U_l)$ to infer as much information about l as possible. This is done by using

$$promote' \;\equiv\; \lambda(U, L).(U, \alpha(map(\lambda l.big\_meet(or\_ext(\lambda z.join(z, l))))(L)))$$

This function, when applied to (U, L) returns the new lower approximation in the form $\langle\{l_1\}, \ldots, \{l_n\}\rangle$ instead of $\{l_1, \ldots, l_n\}$. Thus, the operation *promote* can now be defined as $\lambda(U, L).\mu(or\_to\_set(promote'(U, L)))$.

Applying *promote* to the relations Employees and CS1 gives us the new lower approximation that consists of one record [John, 15K, 76]. Thus, it tells us that John from office 76 is a TA with salary 15K, and Mary with salary 12K could be a TA. Hence the result is an approximation in the sense of Lipski [27, 28]: we have the set of "for sure" answers and the set of "maybe" answers.

6 Conclusion

Previous papers on approximate answers to queries against independent databases ([5, 13, 29, 31]) do not address two important problems, which are required for a general theory. First, we need 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], which proposed the *promote* operation, but no general principles were 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 representing partiality via orders on objects (cf. [5, 21, 22]), we have given formal models of approximate answers to queries and classified them, arriving at ten possible constructs.

- We have explained a new approach to query language design, based on turning universality properties into syntax, thus obtaining the introduction and elimination operations for the data types. To apply this approach to approximations, we need the operations *naturally* associated with them. To find such operations, we have characterized most of the approximation constructs via their universality properties.

  It must be emphasized that, in contrast to datatypes such as sets, bags and lists, finding universality properties for approximation is a nontrivial algebraic problem. Moreover, we have obtained results of a new kind saying that some of the constructs do *not* possess those 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 cannot verify all preconditions for well-definedness. To overcome these problems, we suggested using or-sets to encode approximations, and showed how the language from [15, 25] can be used to answer some typical queries.

  Despite the fact that a straightforward application of the data-oriented approach did not lead to a practical language, we still regard the work on universality of approximations as very useful. After all, those properties gave us the operations naturally associated with the constructs, and enabled us to prove theorem 14 which is the best justification for using $or\mathrm{-}\mathcal{NRA}$ to program with approximations.

A number of open problems remain. For two constructs no universality results are known, and we believe that negative results can be proved. We believe that additional optimizations can be found for standard procedures for querying independent databases. That is, the implementation shown in this paper is not the most efficient one, and this may influence the design of a language that deals with approximations. The last two items are more speculative. First, it may be interesting to see what (if any) are the connections between our work and recent work [10, 11] on approximating recursive datalog programs with nonrecursive ones. Second, we have shown that some modal operations are naturally associated with approximations. Modal operations have been used in the context of incomplete information in databases, for example, by [25, 33] to describe conjunctive and disjunctive sets by means of modal connectives, and in [20] to provide semantics of constraints. Whether there are any connections between [20, 33] and our work, remains to be seen.

Acknowledgements. I wish to thank Peter Buneman, Carl Gunter, Elsa Gunter, Achim Jung, Paris Kanellakis, Hermann Puhlmann, Anna Romanowska, Moshe Vardi and the reviewers for their comments and suggestions, and Tim Griffin for a careful reading of the manuscript.

References

[1] S. Abiteboul, R. Hull and V. Vianu. *"Foundations of Databases"*, Addison Wesley, 1995.

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

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

[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*, pages 60–75. Springer, 1991.

[5] P. Buneman, S. Davidson, A. Watters, A semantics for complex objects and approximate answers, *Journal of Computer and System Sciences* 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] P. Buneman, L. Libkin, D. Suciu, V. Tannen and L. Wong. Comprehension syntax. *SIGMOD Record*, 23 (1994), 87–96.

[8] P. Buneman, S. Naqvi, V. Tannen and L. Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science* 149 (1995), 3–48.

[9] 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.

[10] S. Chaudhuri. Finding nonrecursive envelopes for database predicates. In *ACM Symposium on Principles of Database Systems (PODS'93)*, pages 135–146.

[11] S. Chaudhuri and Ph. Kolaitis. Can Datalog be approximated? In *ACM Symposium on Principles of Database Systems (PODS'94)*, pages 86–96.

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

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

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

[15] E. Gunter and L. Libkin, OR-SML: A functional database programming language for disjunctive information and its applications. *5th Int. Conf. on Database and Expert Systems Applications*, LNCS 856, Springer Verlag, 1994, pages 641–650.

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

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

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

[19] A. Jung and H. Puhlmann, Types, logic, and semantics for nested databases. In: *Proc. of the Conference on Mathematical Foundations of Programming Semantics–95*, Electronic Notes in Theoretical Computer Science, Elsevier Science Publishers, 1995.

[20] K.L. Kwast. A deontic approach to database integrity. *Annals Mathematics and Artificial Intelligence* 9 (1993), 205–238.

[21] 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.

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

[23] L. Libkin. Approximation in databases, In *Proc. of Intl. Conf. on Database Theory*, pages 411–424, LNCS 893, Springer Verlag, 1995.

[24] L. Libkin and L. Wong, On representation and querying incomplete information in databases with bags. *Information Processing Letters*, 56 (4) (1995), 209–214.

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

[26] L. Libkin and L. Wong, Query languages for bags and aggregate functions. *Journal of Computer and System Sciences*, to appear. Extended abstract in *ACM Symposium on Principles of Database Systems (PODS'94)*, pages 155–166.

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

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

[29] 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.

[30] J. Płonka. On distributive quasilattices. *Fundamenta Mathematicae* 60 (1967), 191–200.

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

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

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

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

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