An Algebra for Pomsets *

Extended Abstract

Stéphane Grumbach[1] and Tova Milo[2]

[1] I.N.R.I.A. Rocquencourt, 78153 Le Chesnay, France, Stephane.Grumbach@inria.fr
[2] Dept. Computer Science, Tel-Aviv University, Tel-Aviv, Israel, milo@db.toronto.edu

Abstract. We study languages for manipulating *partially ordered* structures with *duplicates* (e.g. trees, lists). As a general framework, we consider the *pomset* (partially ordered multiset) datatype. We introduce an algebra for *pomsets*, which generalizes traditional algebras for (nested) sets, bags and lists. This paper is motivated by the study of the impact of different language primitives on the expressive power. We show that the use of partially ordered types increases the expressive power significantly. Surprisingly, it turns out that the algebra when restricted to both unordered (bags) and totally ordered (lists) intermediate types, yields the same expressive power as fixpoint logic with counting on relational databases. It therefore constitutes a rather robust class of relational queries. On the other hand, we obtain a characterization of PTIME queries on lists by considering only totally ordered types.

1 Introduction

In the standard approach to database modeling two restrictions are imposed on database relations. (i) The elements are assumed to be un-ordered, and (ii) no duplicates are allowed. The impact of having order on database elements has been studied extensively, and was shown to be significant for the expressive power of query languages. In particular it was shown that fixpoint logic expresses all PTIME functions on ordered inputs [Imm86, Var82]. The impact of allowing duplicates in relations was studied in [GM93, GMK93, LW94]. It was also shown to have a strong effect on the expressive power of query languages, and in particular to result in an increased expressive power. In this paper, we study the interaction between order and duplicates, and the effect that the combined features have on the expressive power of query languages.

Results on the impact of order on the expressive power of languages focus on *ordered domains*. There is another approach to order which was not explored so far, namely the order related to the data types used in the query language, that we call the *internal order*. Observe that many data types allow both multiple occurrences of the same element, and (partial) order on the occurrences (e.g. *lists, trees*). We refer to such data types as *ordered types*. Supporting ordered

* Work supported in part by the Esprit Project BRA FIDE 2. Work was done while the second author was visiting University of Toronto, and supported by the Institute for Robotics and Intelligent Systems.

types in the data model is essential in databases for advanced applications such as CAD/CAM, Software Engineering, and Text Retrieval [SZL93, ACM93].

Previous research considered *ordered domains* and *internal order* as orthogonal issues. For example, the research on the expressive power of languages focused on languages for manipulating inputs of unordered types like sets and bags, and the intermediate data types used in the computation were also unordered [GM93, GMK93, Imm86, Var82]. This was the case even when the domain of the input was totally ordered. On the other hand, ordered types like lists or trees were mainly used when manipulating inputs of type list or tree [Ric92, GW92, SZL93, CRSV94].

The paper focuses on algebraic languages for manipulating ordered types. We consider the *pomset* (partially ordered multiset) datatype, and present an algebra for *pomsets*. The algebra is an extension of the (nested) relational algebra [AB87]. Our aim is to investigate how the combined use of duplicates and (partial) order affects the expressive power of the languages, and increases the complexity.

Surprisingly, it turns out that the additional expressive power gained by using ordered types with duplicates is similar to that gained by using two well studied primitives — *fixpoint* and *counting*. The use of duplicates introduces counting capabilities, and order enables the simulation of fixpoint. We show that the extension of the relational algebra with un-ordered types with duplicates (like bags) and totally ordered types (like lists) has exactly the same expressive power as FO+FP+C [GO93].

To complete previous works that considered the effect of totally ordered domains on the expressive power of the language, we study here the effect of total internal order. We show that the algebra expresses all PTIME queries on lists. Note that another language for list manipulation, expressing PTIME, was presented in [CRSV94]. This language, however, required the use of nested structures. In contrast, our result shows that all PTIME queries can be expressed even without nesting.

In addition to order and duplicates, there is another parameter – *nesting* – that affects the expressiveness and complexity of query languages. Nesting of structures has been extensively studied in the context of unordered types like sets and bags. We study the combined effect of order, duplicates, and nesting. Is was shown in [GM93] that the presence of duplicates significantly affect the expressive power of the nested algebra. In particular, two levels of nesting were proved to be sufficient for expressing all elementary functions. We show that duplicates with order are even more powerful, and one level of nesting is sufficient for expressing all elementary functions.

The paper is organized as follows. In the next section, we review some preliminaries and define the type *pomset*. The algebra for pomsets, PALG, is introduced in Section 3. In Section 4, we study properties of the algebra, and in Sections 5 and 6, we characterize its complexity and expressive power. Finally, in Section 7, we study the power of the nested algebra.

2 Data Types and Queries

In this section, we introduce formally the various data types, and recall the basic framework of databases and queries over data of complex types. Complexity measures are then defined.

2.1 Partially Ordered Multisets, *pomsets*

We assume the existence of an atomic type U, whose domain is an infinite set of constants. We consider tuples of any arity k, $U^k = [U, ..., U]$. To have a uniform representation for sets, bags, lists, trees, and other ordered types, we use a general data type called *pomset* (partially ordered multiset). Intuitively, a pomset can be viewed as a string with a partial order instead of a total order. The following formal definition (essentially the one of Pratt [Pra84]) is based on labelled partial orders.

Definition 2.1 *A* **partial order** *is a pair* $(V, <)$ *where* $<$ *is an irreflexive transitive binary relation on the vertex set* V*. A* **labelled partial order** *(lpo) is a structure* $(V, \Sigma, \mu, <)$*, where* $(V, <)$ *is a partial order, and* $\mu : V \rightarrow \Sigma$ *labels the vertices of* V *with elements of the set* Σ*.*

Two lpo's $(V, \Sigma, \mu, <)$ *and* $(V', \Sigma, \mu', <')$ *over the same set of labels* Σ *are* **isomorphic** *if there exists a bijection* $\tau : V \rightarrow V'$ *such that for all* $u, v \in V$*,* $\mu(u) = \mu'(\tau(u))$*, and* $u < v$ *iff* $\tau(u) <' \tau(v)$*.*

A **pomset** $\langle V, \Sigma, \mu, < \rangle$ *is the isomorphism class of an lpo* $(V, \Sigma, \mu, <)$*. A pomset* $\langle V, \Sigma, \mu, < \rangle$ *is* **finite** *if* V *is finite.*

Two pomsets $\langle V, \Sigma, \mu, < \rangle$ *and* $\langle V', \Sigma', \mu', <' \rangle$ *are* **isomorphic** *if there exist bijections* $\tau : V \rightarrow V'$ *and* $\nu : \Sigma \rightarrow \Sigma'$*, such that for all* $u, v \in V$ *and for all* $a \in \Sigma$*,* $\mu(u) = a$ *iff* $\mu'(\tau(u)) = \nu(a)$*, and* $u < v$ *iff* $\tau(u) <' \tau(v)$*.*

Remarks : Pomsets are only defined up to isomorphism to hide the identities of the elements in V, so that only the cardinality of V counts, leaving Σ as the only important set underlying a pomset [Pra84]. This also enables us to assume for the sake of simplicity, that, in any two pomsets, the corresponding sets of vertices are disjoint. This hypothesis is important for algebraic operations.
Note the difference between lpo's, denoted with "( )", and pomsets, denoted with "$\langle\ \rangle$", and between isomorphisms on lpo's over the same set of labels, and isomorphisms on pomsets. The lpo's are only used to define pomsets, and in the following we shall only consider the latter.

Standard types such as sets, bags, directed acyclic graphs (dags), and lists, are subtypes of the pomset type, and are obtained by restricting the mapping μ and the kind of order in the pomset. For example, sets require μ to be a 1-1 mapping, and no order, bags may have arbitrary μ, but no order, and lists require total order.

If Σ is a collection of objects of **type** T, we denote the type of the pomset $\langle V, \Sigma, \mu, < \rangle$ by $\{\!|T|\!\}$. For a type T, the **domain** of $\{\!|T|\!\}$, is the set of all finite pomsets $\langle V, \Sigma, \mu, < \rangle$ where Σ is a collection of objects of type T. All collection types are defined as usual, assuming *finite* and *homogeneous* collections.

From the implementation view point, a pomset $p = \langle V, \Sigma, \mu, < \rangle$ can be represented by a directed acyclic graph $G = (V, E)$ where each node $v \in V$ has a label in Σ attached to it, (different nodes may have the same label attached to them), and where an edge $(v_1, v_2) \in E$ iff $v_1 < v_2$.[3]

In the graph G, each connected component corresponds to a set of vertices $V' \subseteq V$ that are related to each other by $<$ and not related to other vertices in $V - V'$. We overload the term connected component, and call the (sub)pomset $p' = \langle V', \Sigma, \mu \downarrow_{V'}, <\downarrow_{V' \times V'} \rangle$ corresponding to a connected component V' of the graph G, a *connected component of the pomset* p. A connected component p' of the pomset p may have several isomorphic (with respect to pomset isomorphism) occurrences in p. We say that a connected component **n-belongs** to a pomset if it has exactly n isomorphic occurrences.

We say that a pomset p is *linear* if $<$ defines a total order on V (lists and stacks can be represented by linear pomsets). We say that p is *pseudo-linear* if $<$ defines a total order on each connected component of p.

2.2 Databases and Queries

A **pomset database** is a set of named pomsets. (Following the relational model conventions, we shall sometimes refer to these pomsets as database relations.) A **pomset schema** is an expression $p : T$, where p is a pomset name, and T is a pomset type. An **instance** of p is a pomset of type T. A **database schema** DB is a finite set of pomset schemas with distinct pomset names. An **instance** of DB is a mapping associating a pomset to every pomset schema in DB.

Queries on pomset databases are defined by extending the classical definition of [CH80] for relational queries. A query is a mapping from an input schema **DB** $= \{P_1, ..., P_n\}$ to an output schema **S**$= \{P_0\}$ with a single pomset, mapping instances of the input schema to instances of the output schema. Queries must be computable and generic, i.e. insensitive to isomorphisms on the databases (based on pomset isomorphisms).

Complexity classes of queries are defined straightforwardly by extending the definition for relational queries. We use as complexity measures the time and space used by a Turing machine to produce a standard encoding of the output database, starting from a standard encoding of the input database. The standard encoding of a pomset p is based on some labelled partial order $(V, \Sigma, \mu, <)$, and is similar to that of the binary relations representing μ and $<$. Note that in this encoding, each object in Σ, is repeated as many times as the vertices it labels in V. This encoding may seem inefficient since pomsets can sometimes be encoded more compactly (for example, un-ordered pomsets like bags can be

[3] A compact representation can be obtained by considering a transitively reduced [Moh89] version of the graph.

encoded by simply attaching to elements in Σ the number of vertices they label). Nevertheless, this encoding fits with the situation in real database systems, where duplicates are explicitly stored, sometimes precisely to avoid the cost of duplicate elimination.

We consider the **data complexity** of queries, i.e. the complexity of the evaluation of a query in terms of the size of the input databases. For each Turing complexity class $\mathcal{C}$ there is a corresponding complexity class of queries, which, for simplicity, we also denote by $\mathcal{C}$.

3 An Algebra for Pomsets

While for un-ordered data types like sets and bags, there is an accepted standard set of algebraic operations, this is not the case for ordered types, and different algebras for manipulating lists and trees exist in the literature [Ric92, GW92, SZL93]. Some of these languages share, however, a common property — to be an extensions of the relational algebra, and when restricted to non ordered structures with no duplicates, most of them yield the relational algebra. Following these lines, we present below a uniform algebra for (partially) ordered structures with duplicates, that is a strict extension of the relational algebra. The algebra is designed so that by restricting the order or the number of duplicates, one gets an algebra for manipulating specific kinds of ordered types. For example, when restricted to unordered structures with no duplicates it is the traditional relational algebra. When duplicates are allowed, it is the bag algebra [GM93]. When restricted to total orders, it is an algebra for lists.

To make the results of this paper general and applicable to previously presented languages, we consider a restricted set of operators — the relational algebra operators (adjusted to this context), and a small set of additional operators that are essential for exploiting the order in ordered types.

We first present a simple version of the algebra, that manipulates pomsets of tuples of atomic elements. (Nested pomsets, and the extensions in the algebra needed to enable manipulation of nested structures, are considered in Section 7). We start by presenting the algebraic operators. Next we study their properties, and in particular consider the dependencies of the operations.

The algebra contains a *Tupling* operator $[o_1, \ldots, o_k]$ that builds tuples of type U^k, and *attribute projection* $x.i$ that extracts the i^{th} attribute of a tuple. Singleton pomsets are constructed using the *pomsetting* operator $\{\!\{o\}\!\}$, that builds a pomset of the form $\langle \{v\}, \{o\}, (v, o), \emptyset \rangle$. (Note that since pomsets are defined up to isomorphism on V, the actual value of v is irrelevant). We shall use below the term *object* to denote a vertex with its associated label.

To construct more complex pomsets we use two classical operations on pomsets which have been considered in the literature [Pra84]. The *additive union* (also called *shuffle* in [Pra84]), and *concatenation*. Let $p_1 = \langle V_1, \Sigma, \mu_1, <_1 \rangle$, $p_2 = \langle V_2, \Sigma, \mu_2, <_2 \rangle$ be two pomsets of type $\{\!\{T\}\!\}$. Assume w.l.g. that V_1 and V_2 are disjoint.

- *Additive union,* $\biguplus$: $p_1 \biguplus p_2$ is a pomset containing all the objects in p_1 and p_2, and preserving the order of those objects.
  $p_1 \biguplus p_2 = \langle V_1 \cup V_2, \Sigma, \mu_1 \cup \mu_2, <_1 \cup <_2 \rangle$.
- *Concatenation,* $\oplus$: $p_1 \oplus p_2$ is a pomset containing all the objects in p_1 and p_2, preserving the order of the objects, and additionally making all the objects in p_1 smaller than those of p_2. $p_1 \oplus p_2 = \langle V_1 \cup V_2, \Sigma, \mu_1 \cup \mu_2, <_1 \cup <_2 \cup (V_1 \times V_2) \rangle$.

The operations $\biguplus$ and $\oplus$ are associative and $\biguplus$ is commutative. It turns out that these two operations are not sufficient for building all possible pomsets [Pra84]. This can be done by adding more operations to the language, as further discussed below.

3.1 The Relational and Bag algebras

The first group of operations that we add to the algebra are the standard union, intersection, and subtraction. These relational operations were extended in [GM93] to handle bags. We now further generalize these operations for pomsets. In bags, all the occurrences of objects with the same label are isomorphic, and thus the operations are defined in terms of the number of occurrences of each label. The generalization to pomsets is based on the observation that in pomsets, objects with the same label may differ due to the the relative order of objects. This relative order is captured by the connected components of the pomset. Thus, instead of considering the number of occurrences of individual labels, we define the operators w.r.t. the number of occurrences of the connected components.

Let $p = \langle V, \Sigma, \mu, < \rangle$, $p_1 = \langle V_1, \Sigma, \mu_1, <_1 \rangle$, $p_2 = \langle V_2, \Sigma, \mu_2, <_2 \rangle$ be three pomsets of type $\{T\}$, where $T = [U, ..., U]$. Assume that V_1 and V_2 are disjoint. Let $p' = \langle V', \Sigma, \mu\downarrow_{V'}, <\downarrow_{V' \times V'} \rangle$ be some connected component of p, p_1 or p_2.

- *Union,* $\bigcup$: $p_1 \cup p_2$ is a pomset, s.t. a connected component p' n-belongs to $p_1 \cup p_2$ iff p' r-belongs to p_1 and q-belongs to p_2 and $n = sup(r, q)$.
- *Intersection,* $\bigcap$: $p_1 \cap p_2$ is a pomset, s.t. a connected component p' n-belongs to $p_1 \cap p_2$ iff p' r-belongs to p_1 and q-belongs to p_2 and $n = min(r, q)$.
- *Subtraction,* $-$: $p_1 - p_2$ is a pomset, s.t. a connected component p' n-belongs to $p_1 - p_2$ iff p' r-belongs to p_1 and q-belongs to p_2 and $n = sup(0, r - q)$.

Another group of relational operations that were extended in [GM93] to handle bags are projection (and its more general version MAP), selection, and cartesian product. We now further generalize these operations for pomsets.

- *Restructuring,* MAP: Assume that φ is an algebra expression[4] mapping objects of type T to pomsets of type $\{T'\}$. $\text{MAP}_\varphi(p)$ is a pomset of type $\{T'\}$, obtained by replacing each $v \in V$ in p by the whole pomset $p_v = \varphi(\mu(v))$, preserving order both within and between the p_v's.

[4] φ may contain any algebra operatorincluding MAP and the operators defined below.

This operation is also called in the literature on pomsets the *expansion* operation. (See [Pra84] for a formal definition.)
For instance, $\mathrm{MAP}_{\lambda x\ \{x.1\}\oplus\{x.2\}}$ denotes the pomset obtained by replacing each tuple by two objects corresponding to the first and second attributes of the tuple, and making the first attribute precede the second in the order. As another example, $MAP_{\lambda x\ \{[x.1,x.3]\}}$ denotes the projection of each tuple in the pomset on its first and third arguments. For brevity, we shall denote the map operations when applied on pomset of tuples, projecting the attributes $i_1,\ldots,i_n$ by $\Pi_{i_1,\ldots,i_n}$.

- *Selection,* $\sigma_{\varphi=\varphi'}$: Assume that φ,φ' are algebra expressions. $\sigma_{\varphi=\varphi'}(p)$ is the pomset obtained from D by removing all the vertices $v\in V$ not satisfying the equation $\varphi(\mu(v))=\varphi'(\mu(v))$, and preserving the order of the remaining objects.
- *Cartesian product,* $\times$: Assume $p_1 : \{\!\{T_1\}\!\}$ and $p_2 : \{\!\{T_2\}\!\}$. The objects in $p_1\times p_2$ are obtained from the cartesian product of the objects in p_1 and in p_2. The order on the objects is the lexicographical order.
  $p_1\times p_2=\langle V_1\times V_2,\Sigma^2,\mu_1\times\mu_2,$
  . $\quad\{([x_1,y_1],[x_1,y_2])\mid(y_1,y_2)\in<_2\}\cup\{([x_1,y_1],[x_2,y_2])\mid(x_1,x_2)\in<_1\}\rangle :$
  . $\quad\{\!\{T_1\times T_2\}\!\}$

So far we considered only relational algebra operators. An operator important for bags [GM93] is duplicate elimination. This operator can be generalized to pomsets either by (1) considering connected components of the pomsets, and eliminating isomorphic occurrences of the same component, or (2) by considering individual objects and eliminating multiple occurrences of the same label (note that for unordered pomsets, i.e. bags, the two options are identical).

- *Duplicate elimination:*
  Connected components, ϵ : $\epsilon(p)$ is a pomset with exactly one occurrence of each of the non-isomorphic connected components of p.
  Individual objects, ϵ_n : $\epsilon_n(p)$ is a pomset where all the vertices $v\in V$ that are mapped by μ to the same label collapse to one unique vertex with this label, and where the order on the new objects is the maximal order consistent with that of the sources of the objects. More formally:
  $\epsilon_n(p)=\langle V',\Sigma,\mu',<'\rangle$ where $V'\subseteq V$ is such that $\mu'=\mu\downarrow_{V'}$ is 1-1, and for each $v_1,v_2\in V'$ such that $\mu'(v_1)=a$ and $\mu'(v_2)=b$, $(v_1,v_2)\in<'$ iff there exists $v_1'\in\mu^{-1}(a)$, $v_2'\in\mu^{-1}(b)$ such that $(v_1',v_2')\in<$, and there is no $v_1''\in\mu^{-1}(a)$, $v_2''\in\mu^{-1}(b)$ such that $(v_2'',v_1'')\in<$.

3.2 Special Pomset Operators

To fully exploit the ordered types as a computational device, we need to be able to transform unordered types into partially ordered ones, and vice-versa. The relational and bag operators, do not support this. We introduce two operations which deterministically transform partially ordered pomsets into unordered ones and reciprocally.

- *Order-destroy*, $\oplus^{-1}$: $\oplus^{-1}(p) = \langle V, \Sigma, \mu, \emptyset \rangle$ is an unordered pomset of type $\{\!\!\{T\}\!\!\}$, containing the same objects as p.
- *Listing*, Λ : This operation deterministically maps an **unordered** multi-set (i.e. a pomset with an empty order) to a pseudo-linear pomset. More formally, $\Lambda(\langle V, \Sigma, \mu, \emptyset \rangle) = \langle V, \Sigma, \mu, <' \rangle$, where $<'$ is the minimal partial order such that for every label o in Σ with exactly n_o distinct $v_i \in V$ for which $\mu(v_i) = o$, $v_1 <' v_2 <' \ldots <' v_{n_o}$. Otherwise, if $< \neq \emptyset$, then $\Lambda(\langle V, \Sigma, \mu, < \rangle) = (\langle V, \Sigma, \mu, < \rangle)$.

For instance, $\Lambda(\langle \{v_1, v_2, v_3, v_4, v_5\}, \{1, 2\}, \{(v_1, 1), (v_2, 1), (v_3, 1), (v_4, 2), (v_5, 2)\}, \emptyset \rangle)$ generates the following partial order[5]: $v_1 \leq v_2 \leq v_3$ and $v_4 \leq v_5$.

To further exploit the internal order in a pomset, we define an operator that enables sequential iteration over components of the pomset, according to their relative order.

Iterator, I: The operator I realizes a structural-recursion [BTBN91] on the pomsets. To simplify the presentation, we first explain how it behaves when applied on totally ordered pomsets (i.e. lists). Let $\emptyset$ denote the empty list of type $\{\!\!\{T\}\!\!\}$. Let $y \neq \emptyset$ be a non empty list of type $\{\!\!\{T\}\!\!\}$, let x be a singleton list of type $\{\!\!\{T\}\!\!\}$ (i.e. a list containing one element), and let $f :\rightarrow \{\!\!\{T'\}\!\!\}$, $g : \{\!\!\{T\}\!\!\} \rightarrow \{\!\!\{T'\}\!\!\}$, and $h : \{\!\!\{T\}\!\!\}, \{\!\!\{T'\}\!\!\} \rightarrow \{\!\!\{T'\}\!\!\}$ be three algebra expressions. The iterator $I_{f,g,h}$ is defined as follows:

- $I_{f,g,h}(\emptyset) = f$,
- $I_{f,g,h}(x) = g(x)$,
- $I_{f,g,h}(x \oplus y) = h(x, I_{f,g,h}(y))$.

Thus, the result of the iterator when applied on the empty list is determined by the algebra expression f. The result when applied on a singleton list is defined by g, and the result when applied on a list of length $n > 1$ is defined recursively by first computing the operator on the last $(n-1)$ elements, and then using h to take into account the first element.

We next explain how the operator works when applied on partially ordered pomsets. Since lists are totally ordered, the operator can process all the elements of the list sequentially. Partially ordered pomsets do not have a total order on the individual elements, but the operator can still exploit the relative order of sub-parts of the pomsets. We say that a pomset x is *serial* iff there are two pomsets $x_1, x_2 \neq \emptyset$ such that $x = x_1 \oplus x_2$. A pomset is called *non-serial* otherwise. Every pomset x can be viewed as a concatenation of (one or more) non-serial pomsets $x_1, \ldots, x_n$, such that $x = x_1 \oplus \ldots \oplus x_n$.

Example 3.1 The pomset
$x = \langle \{a, b, c, d, e\}, \{1, 2\}, \{(a, 1), (b, 2), (c, 1), (d, 2), (e, 1)\},$
. $\{(a, b), (b, e), (c, d), (d, e), (a, e), (c, e)\} \rangle$
is the concatenation of the non-serial pomsets

[5] Note that since pomsets are defined up to isomorphism on V, the specific order of the v_i's is irrelevant, and thus the operation is deterministic.

$x_1 = \langle\{a,b,c,d\},\{1,2\},\{(a,1),(b,2),(c,1),(d,2)\},\{(a,b),(c,d)\}\rangle$,
and
$x_2 = \langle\{e\},\{1\},\{(e,1)\},\emptyset\rangle$.

It turns out that the non-serial pomsets from which a pomset is composed of can be computed efficiently.

Proposition 3.1 Every pomset x has a unique decomposition into non-serial pomsets $x_1,\ldots,x_n$, such that $x = x_1 \oplus \ldots \oplus x_n$. This decomposition can be computed in time polynomial in the size of x.

The algorithm uses the graph representation of a pomset. It is omitted for lack of space.

When the iterator is applied to partially ordered pomsets, instead of sequentially processing single elements, it processes sequentially the non-serial pomsets from which the pomset is composed. Thus, for partially ordered pomsets, the iterator is defined by recursion on the serial structure of the pomset. The definition is the same as above, except that now $\emptyset$ denotes the empty pomset, $y \neq \emptyset$ a non empty pomset, and $x \neq \emptyset$ a non-serial pomset.

4 Properties of the Algebra

Let PALG denote the algebra for pomsets containing all the operations as defined above. The algebra PALG is very similar to the relational and bag algebras [GM93]. When restricted to unordered pomsets it is essentially the bag algebra, and if additionally no two vertices are allowed to have the same label (and each operation is followed by duplicate elimination) it yields the relational algebra. We use the notation PALG_θ, to denote the restriction of PALG without the operation θ. We next study the properties and dependencies of the different operations. We first consider the duplicate elimination operator. Note that this operator received much attention in the context of object oriented languages [AK89, BP91]. It was shown in particular that it plays an important role to get complete languages. This operation is also very significant in the context of ordered types. The next propositions shows the limitations of $\mathrm{PALG}_{\epsilon_n}$.

Proposition 4.1 $\mathrm{PALG}_{\epsilon_n}$ can construct only series-parallel pomsets.

Series-parallel pomsets are defined as the smallest class containing singleton pomsets, and closed under parallel ($\uplus$) and series ($\oplus$) composition.

The proof is by induction on the depth of the expressions defining the pomsets. The only delicate operations are $\times$, MAP, σ and the iterator I (omitted).

Note that there are pomsets that are not series-parallel. For example, it was shown in [Moh89] that the N-shaped pomset
$\mathcal{N} = \langle\{a,b,c,d\},\{1\},\{(a,1),(b,1),(c,1),(d,1)\},\{(a,b),(a,d),(c,d)\}\rangle$,
with four objects all labeled with "1", is not series-parallel. It follows that:

Corollary 4.2 Not all pomsets can be constructed in $\mathrm{PALG}_{\epsilon_n}$.

It turns out that non series-parallel pomsets can be constructed using the duplicate elimination operator. For example, the N-shaped pomset described above can be constructed in PALG as follows:

$$\mathcal{N} = \Pi_1(\epsilon_n((\{[1,a]\} \oplus \{[1,b]\}) \biguplus (\{[1,a]\} \oplus \{[1,d]\}) \biguplus (\{[1,c]\} \oplus \{[1,d]\}))).$$

Moreover, PALG can construct **all** pomsets.

Proposition 4.3 All pomsets can be constructed using the operators $[]$, $\{\}$, $\uplus$, $\oplus$, ϵ_n, and Π.

The proof is rather straightforward — the construction of arbitrary pomsets is similar to the construction of the N-shaped pomset above (omitted). It follows that:

Corollary 4.4 $\mathrm{PALG}_{\epsilon_n} \subset \mathrm{PALG}$.

Pomsets can be represented in the algebra using the operations $[]$, $\{\}$, $\uplus$, $\oplus$, ϵ_n, and Π, and symbols from $\Sigma \cup V$. We nevertheless often use in the following the semantical notation for pomsets, in terms of a set of vertices V, a labelling function $\mu : V \to \Sigma$, and a partial order over V.

We next consider the operations that map unordered structures to (partially) ordered structures (listing Λ), and vice versa (order-destroy $\oplus^{-1}$). We show that these two operators cannot be expressed by other operations.

Proposition 4.5 .

- $\mathrm{PALG}_{\Lambda} \subset \mathrm{PALG}$
- $\mathrm{PALG}_{\oplus^{-1}} \subset \mathrm{PALG}$

Proof is omitted for lack of space.

5 The Complexity of the Algebra

We next study the complexity of evaluating queries in the pomset algebra. As in the classical relational case, we are aiming for characterizations of PALG in terms of complexity classes of queries. It turns out that some of the operations are rather expensive, but when appropriately restricted, they become tractable.

5.1 Operations on Connected Components

First consider the four operations $\bigcup, \bigcap, -$, and ϵ, that operate on connected components of pomsets. Note that these operations require a graph isomorphism test. To have a tractable language, these operations must be restricted. Let F be a family of graphs for which graph isomorphism is polynomial, and where the test of membership in F is also polynomial. We restrict the operations so that if the input pomset is not in F, then the result of the operation is the empty pomset. The class of **planar graphs** satisfy these two requirements [HT74, HW74]. We assume in the following that the operations are restricted to pomsets whose (transitively reduced) graph representation is planar. This guarantees tractability of the operations.

5.2 The iterator

It is important to note that a careless use of the iterator may cause exponential explosion of the size of the result. Consider for example the case where: $f = \emptyset$, $g(x) = a$, and $h(x, y) = y \oplus y$. In this case, if l is a list, then $I_{f,g,h}(l)$ is a list of a's of length $2^{|l|}$.

Consider the algebra containing all the operators described above, including the iterator I. Given an algebra expression e, we call the problem of testing whether there exists an input p for which $e(p)$ is of size $hyper(\mid p \mid)$, the *exponential explosion problem*. It can be shown that that:

Theorem 5.1 The exponential explosion problem is undecidable.

It is easy to verify that every expression in the algebra can be computed in some hyperexponential time in the size of its input. The height of the hyperexponential depends on the number of imbricated iterators. The exponential growth is due to the duplicates generated in the computation. We consider below two restrictions on the language that ensure tractability by restricting the number of duplicates being generated. Both restrictions apply to the use of the iterator I.

Bounded Iterator: The first restriction is similar to the bounded fixpoint operator of [Suc93]. The idea is to restrict the iterator such that the size of the intermediate result in each step of the computation is bounded. The bound is defined using an algebra expression b. The bounded iterator $I_{f,g,h|b}$, when applied on a pomset p, behaves exactly like the unbounded operator, except that it ignores intermediate results whose size is bigger than $b(p)$. More formally:

$I_{f,g,h|b}(p) = I_{f,g,h'}(p)$, where $h'(x, y) = h(x, y)$ if $|h(x, y))| < |b(p)|$, and $h'(x, y) = y$ otherwise.

The key observation is that if only bounded iterators are used in the algebra, then $b(p)$ is guaranteed to be of size polynomial in p, and thus all the intermediate results are of size polynomial in the size of the input.

Let PALG(b) be the algebra containing all the operators of PALG (with the above restrictions on operations on connected components), and where the iterator is bounded. The complexity of PALG(b) is considered in the next theorem.

Theorem 5.2 PALG(b) is in PTIME.

Proof. (Sketch) We prove the result by showing that (i) the maximal number of occurrences of each tuple at any step of the evaluation of the query is bounded by some polynomial in the size of the input, and (ii) all the operations can be computed in polynomial time in the size of their input. The proof is by induction on the number of operations in the expression, and uses Proposition 3.1 to guarantee that the iterator can be computed in polynomial time. □

Duplicate Elimination: Another way to restrict the number of duplicates generated in the computation is to enforce the use of the duplicate elimination operator. We say that an iterator $I_{f,g,h}$ is *duplicate free* if f, g and h have the form $\epsilon_n(f')$, $\epsilon_n(g')$ and $\epsilon_n(h')$ resp., for some algebra expressions f', g', h'. Duplicate free iterators cannot cause exponential explosion of the (intermediate) results[6]. Let PALG(ϵ) be the algebra containing all the operators of PALG, (with the above restrictions on operations on connected components), and where only duplicate free iterator is used. It is easy to see that PALG(ϵ) $\subseteq$ PALG(b), and thus is in PTIME.

We shall see in the following a special case where the two restrictions on the iterator yield equivalent languages. It is open whether in the general case the inclusion is strict.

6 The Expressive Power of the Algebra

We next study the expressive power of the pomset algebra. As in the classical relational case, we are aiming for characterizations of expressiveness of PALG in terms of complexity classes of queries. In particular, we show that even unordered inputs with no duplicates can benefit from the use of ordered data types.

6.1 RALG, BALG, and FO+FP+C

To measure the expressiveness gained by the use of ordered types, we compare the expressive power of the pomset algebra to that of the relational algebra (denoted below as RALG) [Ull88], the bag algebra (denoted as BALG) [GM93], and various extensions of first order logic (FO), such as fixpoint logic (FO+FP), and fixpoint logic with counting (FO+FP+C) [GO93].

The relational and bag algebras can be naturally derived from the pomset algebra by restricting it to respectively unordered structures with no duplicates, and unordered structures. We briefly describe below the language FO+FP+C (for more details see [GO93]).

FO+FP+C is a two-sorted logic, with the sorts *domain* (unordered) and *integer* (with a linear order). The semantics is based on finite structures with a finite segment of the integers. For any finite relational structure $\mathcal{A} = (A, R_1, \ldots, R_n)$, the structure *with a finite segment of the integers* $\mathcal{A}^*$ is defined by: $\mathcal{A}^* = \mathcal{A} \cup (\{0, ..., n-1\}, \leq)$, where $n = \mid A \mid$. If $\varphi(x)$ is a formula with a free variable x of sort domain, then $\#_x[\varphi(x)]$ is a term of sort integer, interpreted as the cardinality of $\{a \mid \varphi(a)\}$. FO+FP+C allows the manipulation of numbers from 0 to n^k for some k, by using tuples of integers of the form $[i_1, \ldots, i_k]$ where each i_j varies from 0 to $n-1$. In general, we will omit the exact representation of the numbers and consider directly numbers from 0 to some n^k.

[6] In fact, a less drastic restriction, where ϵ_n is enforced only after the operations $\times$, MAP, and I, and when h has more than one occurrence of the variable y, is sufficient to prevent exponential explosion. But for the results considered in this paper, the above restrictive definition of the duplicate free iterator is sufficient.

6.2 Expressiveness

When comparing the pomset algebra to RALG, BALG, FO+FP, and FO+FP+C, one consider queries over the same kind of inputs. We thus restrict below our attention to PALG queries mapping set inputs to set outputs. We first show that the pomset algebra is more powerful than the relational and bag algebras.

Theorem 6.1 FO = RALG $\subset$ BALG $\subset$ PALG(ϵ) $\subseteq$ PALG(b).

The strict inclusion of the relational algebra in the bag algebra was proved in [GM93]. Clearly the bag algebra is included in the pomset algebra. The fact that the inclusion is strict follows from the fact that the parity query, $EVEN$ (which is not expressible in BALG [GM93, LW94]), is expressible in PALG.

Proposition 6.2 $EVEN \in$ PALG(ϵ).

Proof: Let R be a relation. The parity of the cardinality of R is expressed by:

$$EVEN(R) \equiv I_{\lambda x;\epsilon_n(\emptyset),\lambda x;\epsilon_n(\{\!\!\{a\}\!\!\}),\lambda x\ y;\epsilon_n(\{\!\!\{a\}\!\!\}-y)}(\Lambda(\mathrm{MAP}_{\{\!\!\{[a]\}\!\!\}}(R))) \neq \{\!\!\{a\}\!\!\}.\Box$$

As we show below, the iterator I is very similar to the traditional fixpoint operator. Thus, the fact that FO is strictly included in PALG is not surprising. It turns out however, that PALG is also more expressive than FO+FP.

Theorem 6.3 FO+FP $\subset$ PALG(ϵ) $\subseteq$ PALG(b).

Proof: (Sketch) The inclusion of FO+FP in PALG is proved by induction. (Omitted for lack of space). The strict inclusion follows from the fact that $EVEN$ is not expressible in FO+FP. $\Box$

We proved that fixpoint can be simulated in PALG. We next prove that the *counting* construct can also be simulated. In particular, we show that (the tractable portion of) PALG, when restricted to both unordered types (such as sets and bags) and totally ordered types (such as lists) yields exactly the same expressive power as first order logic augmented with fixpoint and counting. Let PALG(b)$|_{u,t}$ (respectively PALG(ϵ)$|_{u,t}$) be the restriction[7] of PALG(b) (respectively PALG(ϵ)) to unordered and totally ordered pomsets.

Theorem 6.4 FO+FP+C = PALG(ϵ)$|_{u,t}$ = PALG(b)$|_{u,t}$.

Proof: (Sketch) To prove that FO+FP+C $\subseteq$ PALG(ϵ)$|_{u,t}$ we follow the lines of the classical translation from calculus to algebra [Ull88, AB87, BM92]. The second direction, PALG(b)$|_{u,t}$ $\subseteq$ FO+FP+C, is proved by induction on the structure of the algebraic expression $\Box$

[7] This restriction is enforced as follows. Whenever, the output of an algebraic operation is neither of the unordered form, $p = \langle V, \Sigma, \mu, \emptyset\rangle$, nor of the totally ordered form, $p = \langle V, \Sigma, \mu, <\rangle$, where $<$ is a linear order over V, then the output is the empty pomset by definition.

A computational model characterizing FO+FP+C was introduced in [GO93] as a Turing machine with a relational store inspired from [AV91]. There are some tight connections between the computation on the Turing tape of the computational device and the expressions manipulating lists in $\mathrm{PALG}|_{u,t}$.

It is open if the use of general partially ordered types with duplicates further increases the expressiveness of the language. We conjecture that: $\mathrm{PALG(b)}|_{u,t} \subset \mathrm{PALG(b)}$. More generally, the expressive power resulting from non series-parallel pomsets is badly understood. We conjecture that the query *source*, giving the objects with no predecessor, on N-shaped graphs cannot be expressed in PALG.

6.3 Ordered Input/Output

To complete previous works that considered the impact of totally ordered domains on the expressive power of the language [Imm86, Var82], we study here the effect of having a total internal order. We show that PALG(b) expresses all PTIME queries on ordered types with total internal order and duplicates. It is open if the same result holds for the other tractable version of PALG, PALG(ϵ).

Theorem 6.5 For input/output of type list, PALG(b)=PTIME.

We prove the theorem by showing that the iterator I can be used to simulate the computation of a Turing machine.

Another language for list manipulation, expressing PTIME, was presented in [CRSV94]. To express all PTIME queries, this language required the use of nested structures, i.e. lists of tuples of lists. In contrast, our result shows that all PTIME queries can be expressed even when only flat lists are available.

7 Nesting

We next consider ordered types with nesting. Nesting of structures has been extensively studied in the context of unordered types like sets [HS88, HS89, GV90, GV91] and bags [GM93]. In this section, we study the combined effect of order, nesting, and duplicates. Nested pomsets can be used to represent ordered types with components of some ordered type (e.g. lists of trees). In nested pomsets, every object can be a nested pomset or a tuple with attributes that are nested pomsets. It turns out that nesting increases drastically the expressive power of the language. This is due to the operations linking different levels of nesting.

All the PALG operators presented in Section 3 are extended naturally to handle nested pomsets. The algebra has one additional operator. Let $p = \langle V, \Sigma, \mu, < \rangle$ be a pomset of type $\{\!\{T\}\!\}$.

- *Powerset*, $\mathcal{P}$: Intuitively, $\mathcal{P}(p)$ computes the set of all sub-pomsets of p. More formally, $\mathcal{P}(p) = \langle V_\mathcal{P}, \Sigma_\mathcal{P}, \mu_\mathcal{P}, \emptyset \rangle$ is an unordered pomset of type $\{\!\{\{\!\{T\}\!\}\}\!\}$, where $\mu_\mathcal{P}$ is a 1-1 function, and where for every[8] pomset $p' = \langle V', \Sigma, \mu', <' \rangle$,

[8] Note that since pomsets define isomorphism classes of lpo's, two isomorphic lpo's will have only one occurrence in $\mathcal{P}(p)$.

where $V' \subseteq V$, $\mu' = \mu \downarrow_{V'}$, and $<' \subseteq < \downarrow_{V'}$, there exists $v \in V_{\mathcal{P}}$ such that $\mu_{\mathcal{P}}(v) = p'$.

As in the nested relation (bag) algebra, the powerset operator may cause exponential growth of the output, and the resulting language is no longer tractable. We therefore remove the restrictions from the iterator and the other operations and consider the full pomset algebra.

Note that the nested relation (bag) algebra has an additional operator, *set(bag)-destroy*, that is used to destroy one level of set (bag) nesting [AB87, GM93]. This operation can be simulated in PALG using the MAP operator. In particular, $\mathrm{MAP}_{\lambda x;x}$ destroys one level of pomset nesting. For every pomset p of type $\{\!\{\{\!\{T\}\!\}\}\!\}$, $\mathrm{MAP}_{\lambda x;x}(p)$ is a pomset of type $\{\!\{T\}\!\}$ obtained by expanding each $v \in V$ in p into a whole pomset $p_v = \mu(v)$, preserving order both within and between the p_v's.

In the rest of this section, we study the effect that different levels of nesting have on the expressive power of PALG. A complex type can be viewed as a tree with nodes representing the tuple and pomset constructors. The **nesting** of a type T is the maximal number of pomset nodes in a path from the root to a leaf. We denote the algebra when restricted to pomsets with nesting of depth k, PALG^k. The simple algebra investigated in previous sections allows only one level of pomsets, thus corresponds to PALG^1. We next consider PALG^2, one stage higher in the nesting hierarchy.

We study the relationship between the nested pomset algebra and the nested relation and nested bag algebras. We denote the nested relation (bag) algebra when restricted to set nesting of depth k, RALG^k (BALG^k) [HS88, GV91, GM93]. We first compare RALG^2, BALG^2 and PALG^2 restricted to queries over (nested) sets.

Theorem 7.1 $\mathrm{RALG}^2 \subset \mathrm{BALG}^2 \subset \mathrm{PALG}^2$.

The strict inclusion of RALG^2 in BALG^2 was proved in [GM93]. It was also shown there that BALG^2 is included in PSPACE. The strict inclusion BALG^2 in PALG^2 follows from the next proposition. Let $\mathcal{E}$, denotes the set of *elementary* queries, i.e. of hyper-exponential complexity.

Theorem 7.2 $\mathrm{PALG}^2 = \mathcal{E}$, for unnested, unordered inputs/outputs.

Proof: (Sketch) We show that every $hyper(i)$-time bounded query over unnested bags can be expressed in PALG^2. The proof is based on an encoding of the configurations of a Turing machine in a set of 4-ary tuples of type $[\{\!\{U\}\!\}, \{\!\{U\}\!\}, U, U]$, and using the iterator I to simulate the moves of the machine. $\Box$

A similar proof technique leads to the following result.

Theorem 7.3 $\mathrm{PALG}^2 = \mathcal{E}$, for unnested input/output of type list.

The last subject we consider is queries with general partially ordered, nested, input/output.

Theorem 7.4 For every $k \geq 3$, PALG^k expresses exactly all the elementary queries over inputs of nesting $(k-1)$.

The proof is based on an encoding of nested pomsets in nested bags, and using a result from [GM93] (Omitted). Note that a similar result was obtained in [GM93] for BALG^k. It follows that:

Theorem 7.5 $\mathrm{RALG}^k \subset \mathrm{BALG}^k = \mathrm{PALG}^k$, for every $k \geq 3$, and input/output of nesting height $(k-1)$.

8 Conclusion

The pomset data type generalyzes all the classical collection types manipulated in databases. We proposed an algebra for pomsets which extends classical algebras for sets, bags and lists. When restricted to specific types, we obtain nice equivalences with either logic (FO+FP+C) or complexity classes (PTIME, $\mathcal{E}$). This shows the robustness of the algebra. Nevertheless, the precise expressive power of the full pomset datatype (including non series-parallel) is still an open question.

Acknowledgments: The authors wish to thank Vassos Hadzilacos for technical remarks.

References

[AB87] S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex objects. In *Proc. Int. Workshop on Theory and Applications of Nested Relations and Complex Objects (extended abstract)*, Darmstadt, 1987. INRIA research report n 846.

[ACM93] S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *Proc. 19th Int'l Conf. on Very Large Data Bases*, 1993.

[AK89] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 159–173, 1989.

[AV91] S. Abiteboul and V. Vianu. Generic computation and its complexity. In *Proc. ACM Symp. on Theory of Computing*, New Orleans, May 1991.

[BM92] C. Beeri and T. Milo. Functional and predicative programming in oodb's. In *Proc. 11th Symp. on Principles of Database Systems San-Diego*, 1992.

[BP91] J. Van Den Bussche and J. Paredaens. The expressive power of structured values in pure oodb's. In *Proc. 10th Symp. on Principles of Database Systems*, 1991.

[BTBN91] V. Breazu-Tannen, P. Buneman, and S. Naqvi. Structural recursion as a query language. In *Proc. 3rd Int. Workshop on database programming languages*, pages 9–19, Nafplion, Aug. 1991. Morgan Kaufman.

[CH80] A. Chandra and D. Harel. Computable Queries for Relational Data Bases. *Journal of Computer and System Sciences*, 21(2):156–178, Oct. 1980.

[CRSV94] L.S. Colby, E.L. Robertson, L.V. Saxton and D. Van Gucht. A Query Language for List Based Complex-Objects. In *Proc. 13th ACM Symp. on Principles of Database Systems*, pages 179–189, Minneapolis, May 1993.

[GM93] S. Grumbach and T. Milo. Towards tractable algebras for bags. In *Proc. 12th ACM Symp. on Principles of Database Systems*, pages 49–58, Washington, May 1993.

[GMK93] S. Grumbach, T. Milo, and Y. Kornatzky. Calculi for bags and their complexity. In *4th Int. Workshop on Database Programming Languages*, New-York City, Septembre 1993. Morgan Kaufmann.

[GO93] E. Graedel and M. Otto. Inductive definability with counting on finite structures. In *Proc. of Computer Science Logic 92*, pages 231–247. LNCS, n702, 1993.

[GV90] S. Grumbach and V. Vianu. Playing games with objects. In *Proc. Int. Conf. on Database Theory*, pages 25–38, Paris, dec 1990.

[GV91] S. Grumbach and V. Vianu. Tractable query languages for complex object databases. In *Proc. 10th ACM Symp. on Principles of Database Systems*, pages 315–327, Boulder, May 1991.

[GW92] S. Ginsburg and X. Wang. Towards a unified approach to querying sequenced data. In *Proc. 11th ACM Symp. on Principles of Database Systems*, pages 293–300, San Diego, 1992.

[HS88] R. Hull and J. Su. On the expressive power of database queries with intermediate types. In *Proc. 7th ACM Symp. on Principles of Database Systems*, 1988.

[HS89] R. Hull and J. Su. Untyped Sets, Invention and Computable Queries. In *Proc. 8th ACM Symp. on Principles of Database Systems*, 1989.

[HT74] J.E. Hopcroft and R.E. Tarjan. Efficient planarity testing. *J. of the ACM*, 21:549–568, 1974.

[HW74] J.E. Hopcroft and J.K. Wong. Linear time algorithm for isomorphism of planar graphs. In *Proc 6th ACM Symp. on THeory of Computing*, pages 172–184, 1974.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Inf. and Control*, 68:86–104, 1986.

[LW94] L. Libkin, L. Wong. New Techniques for Studying Set Languages, Bag Languages, and Aggregate Functions. In *Proc. 13th ACM Symp. on Principles of Database Systems*, pages 155–166, Minneapolis, May 1993.

[Moh89] R. H. Mohring. *Algorithms and Order, I. Rival Ed.*, chapter Computationally Tractable Classes of ordered Sets, pages 105–193. Kluwer Academic Publishers, 1989.

[Pra84] V. Pratt. The pomset model of parallel processes: Unifying the temporal and the spatial. In *Seminar on Concurrency*, pages 180–196, Pittsburgh, 1984. LNCS 197.

[Ric92] J. Richardson. Supporting lists in a data model (a timely approach). In *Proc. 18th Intl. Conf. on Very Large Databases*, Vancouver, 1992.

[Suc93] D. Suciu. Bounded fixpoints for complex objects. In *4th Int. Workshop on Database Programming Languages*, New-York City, Septembre 1993. Morgan Kaufmann.

[SZL93] B. Subramanian, S. B. Zdonik, and T. W. Leung. Ordered types in the aqua data model. In *4th Int. Workshop on Database Programming Languages*, New-York City, Septembre 1993. Morgan Kaufmann.

[Ull88] J.D. Ullman. *Database and Knowledge Base Systems*. Computer Science Press, 1988.

[Var82] M. Vardi. Relational queries computable in polynomial time. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 137–146, 1982.