Possible and Certain Answers for Queries over Order-Incomplete Data*†

Antoine Amarilli[1], Mouhamadou Lamine Ba[2], Daniel Deutch[3], and Pierre Senellart[4,5]

1 **LTCI, Télécom ParisTech, Université Paris-Saclay, Paris, France**
2 **University Alioune Diop of Bambey, Bambey, Senegal**
3 **Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel**
4 **DI ENS, ENS, CNRS, PSL Research University, Paris, France**
5 **Inria Paris, Paris, France**

Abstract

To combine and query ordered data from multiple sources, one needs to handle uncertainty about the possible orderings. Examples of such "order-incomplete" data include integrated event sequences such as log entries; lists of properties (e.g., hotels and restaurants) ranked by an unknown function reflecting relevance or customer ratings; and documents edited concurrently with an uncertain order on edits. This paper introduces a query language for order-incomplete data, based on the positive relational algebra with order-aware accumulation. We use partial orders to represent order-incomplete data, and study possible and certain answers for queries in this context. We show that these problems are respectively NP-complete and coNP-complete, but identify many tractable cases depending on the query operators or input partial orders.

1998 ACM Subject Classification H.2.1 [Database Management] Logical Design

Keywords and phrases certain answer, possible answer, partial order, uncertain data

Digital Object Identifier 10.4230/LIPIcs.TIME.2017.4

1 Introduction

Many applications need to combine and transform ordered data (e.g., temporal data, rankings, preferences) from multiple sources. Examples include sequences of readings from multiple sensors, or log entries from different applications or machines, that must be combined to form a complete picture of events; rankings of restaurants and hotels published by different websites, their ranking function being often proprietary and unknown; and concurrent edits of shared documents, where the order of contributions made by different users needs to be merged. Even when the order of items from each individual source is known, the order across sources is often *uncertain*. For instance, even when sensor readings or log entries have timestamps, these may be ill-synchronized across sensors or machines; different websites may follow different rules and rank different hotels, so there are multiple ways to create a unified ranked list; concurrent document editions may be ordered in multiple ways. We say that the resulting information is *order-incomplete*.

This paper studies query evaluation over order-incomplete data in a relational setting [1]. Our running example is that of restaurants and hotels from travel websites, ranked according

* An extended version of this article can be found at `https://arxiv.org/abs/1707.07222`.
† This research was partially supported by the Israeli Science Foundation (grant 1636/13) and the Blavatnik ICRC.

to proprietary functions. An example query could compute the union of ranked lists of restaurants from distinct websites, or ask for a ranked list of pairs of a restaurant and a hotel in the same district. As we do not know how the proprietary order is defined, the query result may become *uncertain*: there may be multiple reasonable orderings of restaurants in the union result, or multiple orderings of restaurant–hotel pairs. We also study the application of order-aware *accumulation* to the query result, where each possible order may yield a different value: e.g., extracting only the highest ranked pairs, concatenating their names, or assessing the attractiveness of a district based on its best restaurants and hotels.

Our approach is to handle this uncertainty through the classical notions of *possible and certain answers*. First, whenever there is a *certain answer* to the query – i.e., there is only one possible order on query results or one accumulation result – which is obtained no matter the order on the input and in intermediate results, we should present it to the user, who can then browse through the ordered query results (as is typically done in absence of uncertainty, using constructs such as SQL's `ORDER BY`). Certain answers can arise even in non-trivial cases where the combination of input data admits many possible orders: consider user queries that select only a small interesting subset of the data (for which the ordering happens to be certain), or a short summary obtained through accumulation over large data. In many other cases, the different orders on input data or the uncertainty caused by the query may lead to several *possible answers*. In this case, it is still of interest (and non-trivial) to verify whether an answer is possible, e.g., to check whether a given ranking of hotel–restaurant pairs is consistent with a combination of other rankings (the latter done through a query). Thus, we study the problems of deciding whether a given answer is *certain*, and whether it is *possible*.

We note that users may wish to focus on the position of some tuples of interest (e.g., "is it possible/certain that a particular pair of restaurant-hotel is ranked first?", or "is it possible/certain that restaurant A is ranked above restaurant B?). We show these questions may be expressed in our framework through proper choices of accumulation functions.

Main contributions. We introduce a query language with accumulation for order-incomplete data, which generalizes the positive relational algebra [1] with aggregation as the outermost operation. We define a bag semantics for this language, without assuming that a single choice of order can be made (unlike, e.g., rank aggregation [15]): we use *partial orders* to represent all orders that are consistent with the input data. We then undertake the first general study of the *complexity of possible and certain answers for queries over such data*. We show that these problems are respectively NP-complete and coNP-complete, the main difficulties being the existence of duplicate tuple values in the data and the use of order-aware accumulation. Fortunately, we can show many realistic tractable cases: certainty is in PTIME without accumulation, and both problems are tractable under reasonable restrictions on the input and on the query. The rest of this paper is organized as follows. In Section 2, we introduce our data model and our query language. We define and exemplify the problems of possible and certain answers in Section 3. We then study their complexity, first in the general case (Section 4), then in restricted settings that ensure tractability (Sections 5 and 6). We study extensions to the language, namely duplicate elimination and group-by, in Section 7. We compare our model and results with related work in Section 8, and conclude in Section 9. Proof sketches of some important results are given in an appendix, for lack of space.

2 Data Model and Query Language

We fix a countable set of values $\mathcal{D}$ that includes $\mathbb{N}$ and infinitely many values not in $\mathbb{N}$. A *tuple* t *over* $\mathcal{D}$ of *arity* $\mathrm{a}(t)$ is an element of $\mathcal{D}^{\mathrm{a}(t)}$, denoted $\langle v_1, \ldots, v_{\mathrm{a}(t)} \rangle$. The simplest notion

restname	*distr*
Gagnaire	8
TourArgent	5

(a) *Rest* table

hotelname	*distr*
Mercure	5
Balzac	8
Mercure	12

(b) *Hotel* table

hotelname	*distr*
Balzac	8
Mercure	5
Mercure	12

(c) $Hotel_2$ table

Figure 1 Running example: Paris restaurants and hotels.

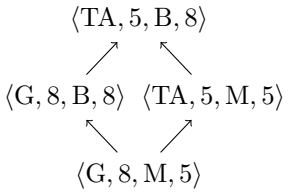

Figure 2 Example 2.

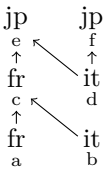

Figure 3 Example 11.

of ordered relations are then *list relations* [11, 12]: a list relation of arity $n \in \mathbb{N}$ is an ordered list of tuples over $\mathcal{D}$ of arity n (where the same tuple value may appear multiple times). List relations impose a single order over tuples, but when one combines (e.g., unions) them, there may be multiple plausible ways to order the results.

We thus introduce *partially ordered relations* (*po-relations*). A po-relation $\Gamma = (ID, T, <)$ of arity $n \in \mathbb{N}$ consists of a finite set of *identifiers* ID (chosen from some infinite set closed under product), a *strict partial order* $<$ on ID, and a (generally non injective) mapping T from ID to $\mathcal{D}^n$. The actual identifiers do not matter, but we need them to refer to occurrences of the same tuple value. Hence, we always consider po-relations *up to isomorphism*, where $(ID, T, <)$ and $(ID', T', <')$ are *isomorphic* iff there is a bijection $\varphi : ID \to ID'$ such that $T'(\varphi(id)) = T(id)$ for all $id \in ID$, and $\varphi(id_1) <' \varphi(id_2)$ iff $id_1 < id_2$ for all $id_1, id_2 \in ID$.

A special case of po-relations are *unordered po-relations* (or *bag relations*), where $<$ is empty: we write them (ID, T). The *underlying bag relation* of $\Gamma = (ID, T, <)$ is (ID, T).

The point of po-relations is to represent *sets* of list relations. Formally, a *linear extension* $<'$ of $<$ is a total order on ID such that for each $x < y$ we have $x <' y$. The *possible worlds* $pw(\Gamma)$ of Γ are then defined as follows: for each linear extension $<'$ of $<$, writing ID as $id_1 <' \cdots <' id_{|ID|}$, the list relation $(T(id_1), \ldots, T(id_{|ID|}))$ is in $pw(\Gamma)$. As T is generally not injective, two different linear extensions may yield the same list relation. Po-relations can thus model uncertainty over the *order* of tuples (but not on their *value*: the underlying bag relation is always certain).

Query language. We now define a bag semantics for *positive relational algebra* operators, to manipulate po-relations with queries. The positive relational algebra, written PosRA, is a standard query language for relational data [1]. We will extend PosRA later in this section

with *accumulation*, and add further extensions in Section 7. Each PosRA operator applies to po-relations and computes a new po-relation; we present them in turn.

The selection operator restricts the relation to a subset of its tuples, and the order is the restriction of the input order. The *tuple predicates* allowed in selections are Boolean combinations of equalities and inequalities, which can use tuple attributes and values in $\mathcal{D}$.

selection: For any po-relation $\Gamma = (ID, T, <)$ and tuple predicate ψ, we define the selection $\sigma_\psi(\Gamma) := (ID', T_{|ID'}, <_{|ID'})$ where $ID' := \{id \in ID \mid \psi(T(id)) \text{ holds}\}$.

The projection operator changes tuple values in the usual way, but keeps the original tuple ordering in the result, and retains all copies of duplicate tuples (following our *bag semantics*):

projection: For a po-relation $\Gamma = (ID, T, <)$ and attributes $A_1, \ldots, A_n$, we define the projection $\Pi_{A_1,\ldots,A_n}(\Gamma) := (ID, T', <)$ where T' maps each $id \in ID$ to $\Pi_{A_1,\ldots,A_n}(T(id))$.

As for union, we impose the minimal order constraints that are compatible with those of the inputs. We use the *parallel composition* [7] of two partial orders $<$ and $<'$ on disjoint sets ID and ID', i.e., the partial order $<'' := (< \parallel <')$ on $ID \cup ID'$ defined by: every $id \in ID$ is incomparable for $<''$ with every $id' \in ID'$; for each $id_1, id_2 \in ID$, we have $id_1 <'' id_2$ iff $id_1 < id_2$; for each $id'_1, id'_2 \in ID'$, we have $id'_1 <'' id'_2$ iff $id'_1 <' id'_2$.

union: Let $\Gamma = (ID, T, <)$ and $\Gamma' = (ID', T', <')$ be two po-relations of the same arity. We assume that the identifiers of Γ' have been renamed if necessary to ensure that ID and ID' are disjoint. We then define $\Gamma \cup \Gamma' := (ID \cup ID', T'', < \parallel <')$, where T'' maps $id \in ID$ to $T(id)$ and $id' \in ID'$ to $T'(id')$.

The union result $\Gamma \cup \Gamma'$ does not depend on the exact definition of Γ'', i.e., it is unique up to isomorphism. Our definition also implies that $\Gamma \cup \Gamma$ is different from Γ, as per bag semantics. In particular, when Γ and Γ' have only one possible world, $\Gamma \cup \Gamma'$ usually does not.

We next introduce two possible product operators. First, the *direct product* [40] $<_{\texttt{DIR}} := (< \times_{\texttt{DIR}} <')$ of two partial orders $<$ and $<'$ on sets ID and ID' is defined by $(id_1, id'_1) <_{\texttt{DIR}} (id_2, id'_2)$ for each $(id_1, id'_1), (id_2, id'_2) \in ID \times ID'$ iff $id_1 < id_2$ and $id'_1 <' id'_2$. We define the direct product operator over po-relations accordingly: two identifiers in the product are comparable only if *both components* of both identifiers compare in the same way.

direct product: For any po-relations $\Gamma = (ID, T, <)$ and $\Gamma' = (ID', T', <')$, remembering that the sets of possible identifiers is closed under product, we let $\Gamma \times_{\texttt{DIR}} \Gamma' := (ID \times ID', T'', < \times_{\texttt{DIR}} <')$, where T'' maps each $(id, id') \in ID \times ID'$ to the *concatenation* $\langle T(id), T'(id') \rangle$.

Again, the direct product result often has multiple possible worlds even when inputs do not.

The second product operator uses the lexicographic product (or *ordinal product* [40]) $<_{\texttt{LEX}} := (< \times_{\texttt{LEX}} <')$ of two partial orders $<$ and $<'$, defined by $(id_1, id'_1) <_{\texttt{LEX}} (id_2, id'_2)$ for all $(id_1, id'_1), (id_2, id'_2) \in ID \times ID'$ iff either $id_1 < id_2$, or $id_1 = id_2$ and $id'_1 <' id'_2$.

lexicographic product: For any po-relations $\Gamma = (ID, T, <)$ and $\Gamma' = (ID', T', <')$, we define $\Gamma \times_{\texttt{LEX}} \Gamma'$ as $(ID \times ID', T'', < \times_{\texttt{LEX}} <')$ with T'' defined like for direct product.

Last, we define the *constant expressions* that we allow:

const:
- for any tuple t, the singleton po-relation $[t]$ has only one tuple with value t;
- for any $n \in \mathbb{N}$, the po-relation $[\leqslant n]$ has arity 1 and has $pw([\leqslant n]) = \{(1), \ldots, (n)\}$.

A natural question is then to determine whether any of our operators is subsumed by the others, but we show that this is not the case:

▶ **Theorem 1.** *No PosRA operator can be expressed through a combination of the others.*

We have now defined a semantics on po-relations for each PosRA operator. We define a *PosRA query* in the expected way, as a query built from these operators and from relation names. Calling *schema* a set $\mathcal{S}$ of relation names and arities, with an attribute name for each position of each relation, we define a *po-database* D as having a po-relation $D[R]$ of the correct arity for each relation name R in $\mathcal{S}$. For a po-database D and a PosRA query Q we denote by $Q(D)$ the po-relation obtained by evaluating Q over D.

▶ **Example 2.** The po-database D in Figure 1 contains information about restaurants and hotels in Paris: each po-relation has a total order (from top to bottom) according to customer ratings from a given travel website, and for brevity we do not represent identifiers.

Let $Q := \mathit{Rest} \times_{\texttt{DIR}} (\sigma_{\mathit{distr} \neq \text{“12”}}(\mathit{Hotel}))$. Its result $Q(D)$ has two possible worlds:
$(\langle \mathrm{G}, 8, \mathrm{M}, 5\rangle, \langle \mathrm{G}, 8, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, 5, \mathrm{M}, 5\rangle, \langle \mathrm{TA}, 5, \mathrm{B}, 8\rangle)$, $(\langle \mathrm{G}, 8, \mathrm{M}, 5\rangle, \langle \mathrm{TA}, 5, \mathrm{M}, 5\rangle, \langle \mathrm{G}, 8, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, 5, \mathrm{B}, 8\rangle)$.
In a sense, these *list relations* of hotel–restaurant pairs are *consistent* with the order in D: we do not know how to order two pairs, except when both the hotel and restaurant compare in the same way. The *po-relation* $Q(D)$ is represented in Figure 2 as a Hasse diagram (ordered from bottom to top), again writing tuple values instead of tuple identifiers for brevity.

Consider now $Q' := \Pi(\sigma_{\mathit{Rest.distr}=\mathit{Hotel.distr}}(Q))$, where Π projects out *Hotel.distr*. The possible worlds of $Q'(D)$ are $(\langle \mathrm{G}, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, \mathrm{M}, 5\rangle)$ and $(\langle \mathrm{TA}, \mathrm{M}, 5\rangle, \langle \mathrm{G}, \mathrm{B}, 8\rangle)$, intuitively reflecting two different opinions on the order of restaurant–hotel pairs in the same district. Defining Q'' similarly to Q' but replacing $\times_{\texttt{DIR}}$ by $\times_{\texttt{LEX}}$ in Q, we have $\mathit{pw}(Q''(D)) = (\langle \mathrm{G}, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, \mathrm{M}, 5\rangle)$.

We conclude by observing that we can efficiently evaluate PosRA queries on po-relations:

▶ **Proposition 3.** *For any fixed PosRA query Q, given a po-database D, we can construct the po-relation $Q(D)$ in polynomial time in the size of D (the polynomial degree depends on Q).*

Accumulation. We now enrich PosRA with order-aware *accumulation* as the outermost operation, inspired by *right accumulation* and *iteration* in list programming, and *aggregation* in relational databases. We fix a *monoid* $(\mathcal{M}, \oplus, \varepsilon)$ for accumulation and define:

▶ **Definition 4.** For $n \in \mathbb{N}$, let $h : \mathcal{D}^n \times \mathbb{N}^* \to \mathcal{M}$ be a function called an arity-n *accumulation map*. We call $\mathrm{accum}_{h,\oplus}$ an *arity-n accumulation operator*; its result $\mathrm{accum}_{h,\oplus}(L)$ on an arity-n list relation $L = (t_1, \ldots, t_n)$ is $h(t_1, 1) \oplus \cdots \oplus h(t_n, n)$, and it is ε on an empty L. For complexity purposes, we *always* require accumulation operators to be *PTIME-evaluable*, i.e., given any list relation L, we can compute $\mathrm{accum}_{h,\oplus}(L)$ in PTIME.

The accumulation operator maps the tuples with h to $\mathcal{M}$, where accumulation is performed with $\oplus$. The map h may use its second argument to take into account the absolute position of tuples in L. In what follows, we omit the arity of accumulation when clear from context.

The PosRA$^{\text{acc}}$ language. We define the language PosRA$^{\text{acc}}$ that contains all queries of the form $Q = \mathrm{accum}_{h,\oplus}(Q')$, where $\mathrm{accum}_{h,\oplus}$ is an accumulation operator and Q' is a PosRA query. The *possible results* of Q on a po-database D, denoted $Q(D)$, is the set of results obtained by applying accumulation to each possible world of $Q'(D)$, namely:

▶ **Definition 5.** For a po-relation Γ, we define: $\mathrm{accum}_{h,\oplus}(\Gamma) := \{\mathrm{accum}_{h,\oplus}(L) \mid L \in \mathit{pw}(\Gamma)\}$.

Of course, accumulation has exactly one result whenever the operator $\text{accum}_{h,\oplus}$ does not depend on the order of input tuples: this covers, e.g., the standard sum, min, max, etc. Hence, we focus on accumulation operators which *depend on the order of tuples* (e.g., defining $\oplus$ as concatenation), so there may be more than one accumulation result:

▶ **Example 6.** As a first example, let $\mathit{Ratings}(\mathit{user}, \mathit{restaurant}, \mathit{rating})$ be an *unordered* po-relation describing the numerical ratings given by users to restaurants, where each user rated each restaurant at most once. Let $\mathit{Relevance}(\mathit{user})$ be a po-relation giving a partially-known ordering of users to indicate the relevance of their reviews. We wish to compute a *total rating* for each restaurant which is given by the sum of its reviews weighted by a PTIME-computable weight function w. Specifically, $w(i)$ gives a nonnegative weight to the rating of the i-th most relevant user. Consider $Q_1 := \text{accum}_{h_1,+}(\sigma_\psi(\mathit{Relevance} \times_{\texttt{LEX}} \mathit{Ratings}))$ where we set $h_1(t, n) := t.\mathit{rating} \times w(n)$, and where ψ is the tuple predicate: $\mathit{restaurant} =$ "Gagnaire" $\wedge$ $\mathit{Ratings.user} = \mathit{Relevance.user}$. The query Q_1 gives the total rating of "Gagnaire", and each possible world of *Relevance* may lead to a different accumulation result.

As a second example, consider an unordered po-relation $\mathit{HotelCity}(\mathit{hotel}, \mathit{city})$ indicating in which city each hotel is located, and consider a po-relation $\mathit{City}(\mathit{city})$ which is (partially) ranked by a criterion such as interest level, proximity, etc. Now consider the query $Q_2 := \text{accum}_{h_2,\text{concat}}(\Pi_{\mathit{hotel}}(Q_2'))$, where $Q_2' := \sigma_{\mathit{City.city}=\mathit{HotelCity.city}}(\mathit{City} \times_{\texttt{LEX}} \mathit{HotelCity})$, where $h_2(t, n) := t$, and where "concat" denotes standard string concatenation. Q_2 concatenates the hotel names according to the preference order on the city where they are located, allowing any possible order between hotels of the same city and between hotels in incomparable cities.

3 Possibility and Certainty

Evaluating a PosRA or PosRA$^{\text{acc}}$ query Q on a po-database D yields a *set of possible results*: for PosRA$^{\text{acc}}$, it yields an explicit set of accumulation results, and for PosRA, it yields a po-relation that represents a set of possible worlds (list relations). The uncertainty among the results may be due to the order of the input relations being partial, due to uncertainty yielded by the query, or both. In some cases, there is only one possible result, i.e., a *certain* answer. In other cases, we may wish to examine multiple *possible* answers. We thus define:

▶ **Definition 7** (Possibility and Certainty)**.** Let Q be a PosRA query, D be a po-database, and L a list relation. The *possibility problem* (POSS) asks if $L \in pw(Q(D))$, i.e., if L is a possible result. The *certainty problem* (CERT) asks if $pw(Q(D)) = \{L\}$, i.e., if L is the only possible result. Likewise, if Q is a PosRA$^{\text{acc}}$ query with accumulation monoid $\mathcal{M}$, for a result $v \in \mathcal{M}$, the POSS problem asks whether $v \in Q(D)$, and CERT asks whether $Q(D) = \{v\}$.

Discussion. For PosRA$^{\text{acc}}$, our definition follows the usual notion of possible and certain answers in data integration [28] and incomplete information [30]. For PosRA, we ask for possibility or certainty of an *entire* output list relation, i.e., *instance possibility and certainty* [3]. We now justify that these notions are useful and discuss more "local" alternatives.

First, as we exemplify below, the output of a query may be certain even for complex queries and uncertain input. It is important to identify such cases and present the user with the certain answer in full, like order-by query results in current DBMSs. Our CERT problem is useful for this task, because we can use it to decide if a certain output exists, and if yes, we can compute it in PTIME (by choosing any linear extension). However, CERT is a challenging problem to solve, because of duplicate values (see "Technical difficulties" below).

▶ **Example 8.** Consider the po-database D of Figure 1 with the po-relations *Rest* and $Hotel_2$. To find recommended pairs of hotels and restaurants in the same district, the user can write $Q := \sigma_{Rest.distr=Hotel_2.distr}(Rest \times_{\texttt{DIR}} Hotel_2)$. Evaluating $Q(D)$ yields only one possible world, namely, the list relation $(\langle G, 8, B, 8\rangle, \langle TA, 5, M, 5\rangle)$, which is a *certain* result.

This could also happen with larger input relations. Imagine for example that we join hotels and restaurants to find pairs of a hotel and a restaurant located in that hotel. The result can be certain if the relative ranking of the hotels and of their restaurants agree.

If there is no certain answer, deciding possibility of an instance may be considered as "best effort". It can be useful, e.g., to check if a list relation (obtained from another source) is consistent with a query result. For example, we may wish to check if a website's ranking of hotel–restaurant pairs is *consistent* with the preferences expressed in its rankings for hotels and restaurants, to detect when a pair is ranked higher than its components would warrant.

When there is no overall certain answer, or when we want to check the possibility of some aggregate property of the relation, we can use a PosRA$^{\text{acc}}$ query. In particular, in addition to the applications of Example 6, accumulation allows us to encode alternative notions of `POSS` and `CERT` for *PosRA* queries, and to express them as `POSS` and `CERT` for PosRA$^{\text{acc}}$. For example, instead of possibility or certainty for a full relation, we can express possibility or certainty of the *location*[1] of particular tuples of interest:

▶ **Example 9.** With accumulation we can model *position-based selection* queries. Consider for instance a *top-k* operator on list relations, which retrieves a list relation of the first k tuples. For a po-relation, the set of results is all possible such list relations. We can implement *top-k* as $\text{accum}_{h_3,\text{concat}}$ with $h_3(t, n)$ being (t) for $n \leqslant k$ and ε otherwise, and with concat being list concatenation. We can similarly compute *select-at-k*, i.e., return the tuple at position k, via $\text{accum}_{h_4,\text{concat}}$ with $h_4(t, n)$ being (t) for $n = k$ and ε otherwise.

Accumulation can also be used for a *tuple-level comparison*. To check whether the first occurrence of a tuple t_1 precedes any occurrence of t_2, we define h_5 for all $n \in \mathbb{N}$ by $h_5(t_1, n) := \top$, $h_5(t_2, n) := \bot$ and $h_5(t, n) := \varepsilon$ for $t \neq t_1, t_2$, and a monoid operator $\oplus$ such that $\top \oplus \top = \top \oplus \bot = \top$, $\bot \oplus \bot = \bot \oplus \top = \bot$: the result is ε if neither t_1 not t_2 is present, $\top$ if the first occurrence of t_1 precedes any occurrence of t_2, $\bot$ otherwise.

We study the complexity of these variants in Section 6. We now give examples of their use:

▶ **Example 10.** Consider $Q = \Pi_{distr}(\sigma_{Rest.distr=Hotel.distr}(Rest \times_{\texttt{DIR}} Hotel))$, which computes ordered recommendations of districts including both hotels and restaurants. Using accumulation as in Example 9, the user can compute the best district to stay in with $Q' = \text{top-1}(Q)$. If Q' has a certain answer, then there is a dominating hotel–restaurant pair in this district, which answers the user's need. If there is no certain answer, `POSS` allows the user to determine the *possible* top-1 districts.

We can also use `POSS` and `CERT` for PosRA$^{\text{acc}}$ queries to restrict attention to *tuples* of interest. If the user hesitates between districts 5 and 6, they can apply tuple-level comparison to see whether the best pair of district 5 may be better (or is always better) than that of 6.

Technical difficulties. The main challenge to solve `POSS` and `CERT` for a PosRA query Q on an input po-database D is that the tuple values of the desired result L may occur multiple times in the po-relation $Q(D)$, making it hard to match L and $Q(D)$. In other words, even

[1] Remember that the *existence* of a tuple is not order-dependent and thus vacuous in our setting.

though we may compute the po-relation $Q(D)$ in PTIME (by Proposition 3) and present it to the user, they still cannot easily "read" possible and certain answers out of the po-relation:

▶ **Example 11.** Consider a po-relation $\Gamma = (ID, T, <)$ with $ID = \{id_\text{a}, id_\text{b}, id_\text{c}, id_\text{d}, id_\text{e}, id_\text{f}\}$, with $T(id_\text{a}) := \langle \text{Gagnaire}, \text{fr}\rangle$, $T(id_\text{b}) := \langle \text{Italia}, \text{it}\rangle$, $T(id_\text{c}) := \langle \text{TourArgent}, \text{fr}\rangle$, $T(id_\text{d}) := \langle \text{Verdi}, \text{it}\rangle$, $T(id_\text{e}) := \langle \text{Tsukizi}, \text{jp}\rangle$, $T(id_\text{f}) := \langle \text{Sola}, \text{jp}\rangle$, and with $id_\text{a} < id_\text{c}$, $id_\text{b} < id_\text{c}$, $id_\text{c} < id_\text{e}$, $id_\text{d} < id_\text{e}$, and $id_\text{d} < id_\text{f}$. Intuitively, Γ describes a preference relation over restaurants, with their name and the type of their cuisine. Consider the PosRA query $Q := \Pi(\Gamma)$ that projects Γ on type; we illustrate the result (with the original identifiers) in Figure 3. Let L be the list relation $(\text{it}, \text{fr}, \text{jp}, \text{it}, \text{fr}, \text{jp})$, and consider POSS for Q, Γ, and L.

We have that $L \in pw(Q(\Gamma))$, as shown by the linear extension $id_\text{d} <' id_\text{a} <' id_\text{f} <' id_\text{b} <' id_\text{c} <' id_\text{e}$ of $<$. However, this is hard to see, because each of it, fr, jp appears more than once in the candidate list as well as in the po-relation; there are thus multiple ways to "map" the elements of the candidate list to those of the po-relation, and only some of these mappings lead to the existence of a corresponding linear extension. It is also challenging to check if L is a certain answer: here, it is not, as there are other possible answers, e.g.: $(\text{it}, \text{fr}, \text{fr}, \text{it}, \text{jp}, \text{jp})$.

For PosRA$^\text{acc}$ queries, this technical difficulty is even accrued because of the need to figure out the possible ways in which the desired accumulation result can be obtained.

4 General Complexity Results

We have defined the PosRA and PosRA$^\text{acc}$ query languages, and defined and motivated the problems POSS and CERT. We now start the study of their complexity, which is the main technical contribution of our paper. We will always study their *data complexity*[2], where the query Q is fixed: in particular, for PosRA$^\text{acc}$, the accumulation map and monoid, which we assumed to be PTIME-evaluable, is fixed as part of the query, though it is allowed to be infinite. The input to POSS and CERT for the fixed query Q is the po-database D and the candidate result (a list relation for PosRA, an accumulation result for PosRA$^\text{acc}$).

Possibility. We start with POSS, which we show to be NP-complete in general.

▶ **Theorem 12.** *The POSS problem is in NP for any PosRA or PosRA$^\text{acc}$ query. Further, there exists a PosRA query and a PosRA$^\text{acc}$ query for which the POSS problem is NP-complete.*

In fact, as we will later point out, hardness holds even for quite a restrictive setting, with a more intricate proof: see Theorem 18.

Certainty. We show that CERT is coNP-complete for PosRA$^\text{acc}$:

▶ **Theorem 13.** *The CERT problem is in coNP for any PosRA$^\text{acc}$ query, and there is a PosRA$^\text{acc}$ query for which it is coNP-complete.*

For PosRA queries, however, we show that CERT is in PTIME. As we will see later, this follows from the tractability of CERT for PosRA$^\text{acc}$ on *cancellative monoids* (Theorem 26).

▶ **Theorem 14.** *CERT is in PTIME for any PosRA query.*

We next identify further tractable cases, first for PosRA and then for PosRA$^\text{acc}$.

[2] In *combined complexity*, with Q part of the input, POSS and CERT are easily seen to be respectively NP-hard and coNP-hard, by reducing from the evaluation of Boolean conjunctive queries (which is NP-hard in data complexity [1]) even without order.

5 Tractable Cases for POSS on PosRA Queries

We show that POSS is tractable for PosRA queries if we restrict the allowed operators and if we bound some order-theoretic parameters of the input po-database, such as *poset width*.

We call $\text{PosRA}_{\text{LEX}}$ the fragment of PosRA that disallows the $\times_{\text{DIR}}$ operator, but allows all other operators (including $\times_{\text{LEX}}$). We also define $\text{PosRA}_{\text{DIR}}$ that disallows $\times_{\text{LEX}}$ but not $\times_{\text{DIR}}$.

Totally ordered inputs. We start by the natural case where the individual po-relations are *totally ordered*, i.e., their order relation is a total order (so they actually represent a list relation). This applies to situations where we integrate data from multiple sources that are certain (totally ordered), and where uncertainty only results from the integration query (so that the result may still have exponentially many possible worlds, e.g., the *union* of two total orders has exponentially many possible interleavings). In a sense, the $\times_{\text{DIR}}$ operator is the one introducing the most uncertainty and "complexity" in the result, so we consider the fragment $\text{PosRA}_{\text{LEX}}$ of PosRA queries without $\times_{\text{DIR}}$, and show:

▶ **Theorem 15.** *POSS is in PTIME for $\text{PosRA}_{\text{LEX}}$ queries if input po-relations are totally ordered.*

In fact, we can show tractability for relations of bounded *poset width*:

▶ **Definition 16** ([36])**.** An *antichain* in a po-relation $\Gamma = (ID, T, <)$ is a set $A \subseteq ID$ of pairwise incomparable tuple identifiers. The *width* of Γ is the size of its largest antichain. The *width* of a po-database is the maximal width of its po-relations.

In particular, totally ordered po-relations have width 1, and unordered po-relations have a width equal to their size (number of tuples); the width of a po-relation can be computed in PTIME [18]. Po-relations of low width are a common practical case: they cover, for instance, po-relations that are totally ordered except for a few tied identifiers at each level. We show:

▶ **Theorem 17.** *For any fixed $k \in \mathbb{N}$ and fixed $\text{PosRA}_{\text{LEX}}$ query Q, the POSS problem for Q is in PTIME when all po-relations of the input po-database have width $\leqslant k$.*

We last justify our choice of disallowing the $\times_{\text{DIR}}$ product. Indeed, if we allow $\times_{\text{DIR}}$, then POSS is hard on totally ordered po-relations, even if we disallow $\times_{\text{LEX}}$:

▶ **Theorem 18.** *There is a $\text{PosRA}_{\text{DIR}}$ query for which the POSS problem is NP-complete even when the input po-database is restricted to consist only of totally ordered po-relations.*

Unordered inputs We now show the tractability of POSS for *unordered* input relations, i.e., po-relations that allow all possible orderings over their tuples. This applies, e.g., to contexts where the order on input tuples is irrelevant or unknown; all order information must then be imposed by the (fixed) query, using the ordered constant relations $[\leqslant_\bullet]$. We show:

▶ **Theorem 19.** *POSS is in PTIME for any PosRA query if input po-relations are unordered.*

Here again we prove a more general result, capturing the case where the input is "almost unordered". We introduce for this purpose a novel order-theoretic notion, *ia-width*, which decomposes the relation in classes of indistinguishable sets of incomparable elements.

▶ **Definition 20.** Given a poset $(V, <)$, a subset $S \subseteq V$ is an *indistinguishable antichain* if it is both an antichain (there are no $x, y \in S$ such that $x < y$) and an *indistinguishable set* (or *interval* [17]): for all $x, y \in S$ and $z \in V \backslash S$, $x < z$ iff $y < z$, and $z < x$ iff $z < y$.

An *indistinguishable antichain partition* (ia-partition) of a poset is a partition of its domain into indistinguishable antichains. The *cardinality* of such a partition is its number of classes. The *ia-width* of a poset (or po-relation) is the cardinality of its smallest ia-partition. The ia-width of a po-database is the maximal ia-width of its relations.

Hence, any po-relation Γ has ia-width at most $|\Gamma|$, and unordered relations have an ia-width of 1. Po-relations may have low ia-width in practice if order is completely unknown except for a few comparability pairs given by users, or when objects of a constant number of types are ordered based only on some order on the types. We show that ia-width, like width, can be computed in PTIME, and that bounding it ensures tractability (for all PosRA):

▶ **Proposition 21.** *The ia-width of any poset, and a corresponding ia-partition, can be computed in PTIME.*

▶ **Theorem 22.** *For any fixed $k \in \mathbb{N}$ and fixed PosRA query Q, the* POSS *problem for Q is in PTIME when all po-relations of the input po-database have ia-width $\leqslant k$.*

Mixing both kinds of relations. We have shown the tractability of POSS assuming constant width (only for $\text{PosRA}_{\text{LEX}}$ queries) or assuming constant ia-width. A natural question is then whether we can allow both *totally ordered* and *unordered* po-relations. For instance, we may combine sources whose order is fully unknown or irrelevant, with sources that are completely ordered (or almost totally ordered). More generally, can we allow both bounded-width and bounded-ia-width relations? We show that this is the case if we disallow *both* product operators, i.e., restrict to the language $\text{PosRA}_{\text{no}\times}$ of PosRA queries with no product.

▶ **Theorem 23.** *For any fixed $k \in \mathbb{N}$ and fixed $\text{PosRA}_{\text{no}\times}$ query Q, the* POSS *problem for Q is in PTIME when all po-relations of the input po-database have either ia-width $\leqslant k$ or width $\leqslant k$.*

Disallowing product is severe, but we can still integrate sources by taking the *union* of their tuples, selecting subsets, and modifying tuple values with projection. In fact, allowing product makes POSS intractable when allowing both unordered and totally ordered input:

▶ **Theorem 24.** *There is a $\text{PosRA}_{\text{LEX}}$ query and a $\text{PosRA}_{\text{DIR}}$ query for which the* POSS *problem is NP-complete even when the input po-database is restricted to consist only of one totally ordered and one unordered po-relation.*

6 Tractable Cases for Accumulation Queries

We next study tractable cases for POSS and CERT in presence of accumulation.

Cancellative monoids. We first consider a natural restriction on the accumulation function:

▶ **Definition 25** ([23])**.** For any monoid $(\mathcal{M}, \oplus, \varepsilon)$, we call $a \in \mathcal{M}$ *cancellable* if, for all $b, c \in \mathcal{M}$, we have that $a \oplus b = a \oplus c$ implies $b = c$, and we also have that $b \oplus a = c \oplus a$ implies $b = c$. We call $\mathcal{M}$ a *cancellative monoid* if all its elements are cancellable.

Many interesting monoids are cancellative; in particular, this is the case of both monoids in Example 6. More generally, all *groups* are cancellative monoids (but some infinite cancellative monoids are not groups, e.g., the monoid of concatenation). For this large class of accumulation functions, we design an efficient algorithm for certainty.

▶ **Theorem 26.** *CERT is in PTIME for any PosRA*$^{\text{acc}}$ *query that performs accumulation in a cancellative monoid.*

Hence, CERT is tractable for PosRA (Theorem 14), via the concatenation monoid, and CERT is also tractable for top-k (defined in Example 9). The hardness of POSS for PosRA (Theorem 12) then implies that POSS, unlike CERT, is hard even on cancellative monoids.

Other restrictions on accumulation. We next revisit the results of Section 5 for PosRA$^{\text{acc}}$. However, we need to make other assumptions on accumulation (besides PTIME-evaluability). First, in the next results in this section, we assume that the accumulation monoid is *finite*:

▶ **Definition 27.** A PosRA$^{\text{acc}}$ query is said to perform *finite* accumulation if the accumulation monoid $(\mathcal{M}, \oplus, \varepsilon)$ is finite.

For instance, if the domain of the output is assumed to be fixed (e.g., ratings in $\{1, \ldots, 10\}$), then select-at-k and top-k (the latter for fixed k), as defined in Example 9, are finite.

Second, for some of the next results, we require *position-invariant accumulation*, namely, that the accumulation map does not depend on the absolute position of tuples:

▶ **Definition 28.** Recall that the accumulation map h has in general two inputs: a tuple and its position. A PosRA$^{\text{acc}}$ query is said to be *position-invariant* if its accumulation map ignores the second input, so that effectively its only input is the tuple itself.

Note that accumulation in the monoid is still performed in order, so we can still perform, e.g., concatenation. These two restrictions do not suffice to make POSS and CERT tractable, but we will use them to lift the results of Section 5.

Revisiting Section 5. We now extend our previous results to queries with accumulation, for POSS and CERT, under the additional assumptions on accumulation that we presented. We call PosRA$^{\text{acc}}_{\text{LEX}}$ and PosRA$^{\text{acc}}_{\text{no}\times}$ the extension of PosRA$_{\text{LEX}}$ and PosRA$_{\text{no}\times}$ with accumulation.

We can first generalize Theorem 17 to PosRA$^{\text{acc}}_{\text{LEX}}$ queries with *finite* accumulation:

▶ **Theorem 29.** *For any PosRA*$^{\text{acc}}_{\text{LEX}}$ *query performing* finite *accumulation, POSS and CERT are in PTIME on po-databases of bounded width.*

We then extend Theorem 22 to PosRA$^{\text{acc}}$ with *finite* and *position-invariant* accumulation:

▶ **Theorem 30.** *For any PosRA*$^{\text{acc}}$ *query performing* finite *and* position-invariant *accumulation, POSS and CERT are in PTIME on po-databases of bounded ia-width.*

Last, we can adapt the tractability result for queries without product (Theorem 23):

▶ **Theorem 31.** *For any PosRA*$^{\text{acc}}_{\text{no}\times}$ *query performing* finite *and* position-invariant *accumulation, POSS and CERT are in PTIME on po-databases whose relations have either bounded width or bounded ia-width.*

The finiteness assumption is important, as the previous result does not hold otherwise. Specifically, there exists a query that performs *position-invariant* but not *finite* accumulation, for which POSS is NP-hard even on unordered po-relations.

Other definitions. Finally, recall that we can use accumulation as in Example 9 to capture *position-based selection* (top-k, select-at-k) and *tuple-level comparison* (whether the first occurrence of a tuple precedes all occurrences of another tuple) for PosRA queries. Using a direct construction for these problems, we can show that they are tractable:

▶ **Proposition 32.** *For any PosRA query Q, the following problems are in PTIME:*

select-at-k: *Given a po-database D, tuple value t, and position $k \in \mathbb{N}$, whether it is possible/certain that $Q(D)$ has value t at position k;*

top-k: *For any* fixed *$k \in \mathbb{N}$, given a po-database D and list relation L of length k, whether it is possible/certain that the top-k values in $Q(D)$ are exactly L;*

tuple-level comparison: *Given a po-database D and two tuple values t_1 and t_2, whether it is possible/certain that the first occurrence of t_1 precedes all occurrences of t_2.*

7 Extensions

We next briefly consider two extensions to our model: group-by and duplicate elimination.

Group-by. First, we extend accumulation with a *group-by* operator, inspired by SQL.

▶ **Definition 33.** Let $(\mathcal{M}, \oplus, \varepsilon)$ be a monoid and $h : \mathcal{D}^k \to \mathcal{M}$ be an accumulation map (cf. Definition 4), and let $\mathbf{A} = A_1, ..., A_n$ be a sequence of attributes: we call $\text{accumGroupBy}_{h,\oplus,\mathbf{A}}$ an *accumulation operator with group-by*. Letting L be a list relation with compatible schema, we define $\text{accumGroupBy}_{h,\oplus,\mathbf{A}}(L)$ as an *unordered* relation that has, for each tuple value $t \in \pi_{\mathbf{A}}(L)$, one tuple $\langle t, v_t \rangle$ where v_t is $\text{accum}_{h,\oplus}(\sigma_{A_1=t.A_1,...A_n=t.A_n}(L))$ with π and σ on the list relation L having the expected semantics. The result on a po-relation Γ is the set of unordered relations $\{\text{accumGroupBy}_{h,\oplus,\mathbf{A}}(L) \mid L \in pw(\Gamma)\}$.

In other words, the operator "groups by" the values of $A_1, ..., A_n$, and performs accumulation within each group, forgetting the order across groups. As for standard accumulation, we only allow group-by as an outermost operation, calling $\text{PosRA}^{\text{accGBy}}$ the language of PosRA queries followed by one accumulation operator with group-by. Note that the set of possible results is generally not a po-relation, because the underlying bag relation is not certain.

We next study the complexity of POSS and CERT for $\text{PosRA}^{\text{accGBy}}$ queries. Of course, whenever POSS and CERT are hard for some $\text{PosRA}^{\text{acc}}$ query Q on some kind of input po-relations, then there is a corresponding $\text{PosRA}^{\text{accGBy}}$ query for which hardness also holds (with empty $\mathbf{A}$). The main point of this section is to show that the converse is not true: the addition of group-by increases complexity. Specifically, we show that the POSS problem for $\text{PosRA}^{\text{accGBy}}$ is hard even on totally ordered po-relations and without the $\times_{\text{DIR}}$ operator:

▶ **Theorem 34.** *There is a $\text{PosRA}^{\text{accGBy}}$ query Q with finite and position-invariant accumulation, not using $\times_{DIR}$, such that POSS for Q is NP-hard even on totally ordered po-relations.*

This result contrasts with the tractability of POSS for $\text{PosRA}_{\text{LEX}}$ queries (Theorem 15) and for $\text{PosRA}^{\text{acc}}_{\text{LEX}}$ queries with finite accumulation (Theorem 29) on totally ordered po-relations.

By contrast, it is not hard to see that the CERT problem for $\text{PosRA}^{\text{accGBy}}$ reduces to CERT for the same query without group-by, so it is no harder than the latter problem. Specifically:

▶ **Theorem 35.** *All CERT tractability results from Section 6 extend to $\text{PosRA}^{\text{accGBy}}$ when imposing the same restrictions on query operators, accumulation, and input po-relations.*

Duplicate elimination. We last study the problem of consolidating tuples with *duplicate values*. To this end, we define a new operator, dupElim, and introduce a semantics for it. The main problem is that tuples with the same values may be ordered differently relative to other tuples. To mitigate this, we introduce the notion of *id-sets*:

▶ **Definition 36.** Given a totally ordered po-relation $(ID, T, <)$, a subset ID' of ID is an *indistinguishable duplicate set* (or *id-set*) if for every $id_1, id_2 \in ID'$, we have $T(id_1) = T(id_2)$, and for every $id \in ID \backslash ID'$, we have $id < id_1$ iff $id < id_2$, and $id_1 < id$ iff $id_2 < id$.

▶ **Example 37.** Consider the totally ordered relation $\Gamma_1 := \Pi_{hotelname}(Hotel)$, with *Hotel* as in Figure 1. The two "Mercure" tuples are not an id-set: they disagree on their ordering with "Balzac". Consider now a totally ordered relation $\Gamma_2 = (ID_2, T_2, <_2)$ whose only possible world is a list relation (A, B, B, C) for some tuples A, B, and C over $\mathcal{D}$. The set $\{id \in ID_2 \mid T_2(id) = B\}$ is an id-set in Γ_2. Note that a singleton is always an id-set.

We define a semantics for dupElim on a totally ordered po-relation $\Gamma = (ID, T, <)$ via id-sets. First, check that for every tuple value t in the image of T, the set $\{id \in ID \mid T(id) = t\}$ is an id-set in Γ. If this holds, we call Γ *safe*, and set dupElim(Γ) to be the singleton $\{L\}$ of the only possible world of the restriction of Γ obtained by picking one representative element per id-set (clearly L does not depend on the chosen representatives). Otherwise, we call Γ *unsafe* and say that duplicate consolidation has *failed*; we then set dupElim(Γ) to be an empty set of possible worlds. Intuitively, duplicate consolidation tries to reconcile (or "synchronize") order constraints for tuples with the same values, and fails when it cannot be done.

▶ **Example 38.** In Example 37, we have $\text{dupElim}(\Gamma_1) = \emptyset$ but $\text{dupElim}(\Gamma_2) = (A, B, C)$.

We then extend dupElim to po-relations by considering all possible results of duplicate elimination on the possible worlds, ignoring the unsafe possible worlds. If no possible worlds are safe, then we *completely fail*:

▶ **Definition 39.** For each list relation L, we let Γ_L be a po-relation such that $pw(\Gamma_L) = \{L\}$. Letting Γ be a po-relation, we set $\text{dupElim}(\Gamma) := \bigcup_{L \in pw(\Gamma)} \text{dupElim}(\Gamma_L)$. We say that dupElim($\Gamma$) *completely fails* if $\text{dupElim}(\Gamma) = \emptyset$, i.e., $\text{dupElim}(\Gamma_L) = \emptyset$ for every $L \in pw(\Gamma)$.

▶ **Example 40.** Consider the totally ordered po-relation *Rest* from Figure 1, and a totally ordered po-relation $Rest_2$ whose only possible world is (Tsukizi, Gagnaire). Consider $Q := \text{dupElim}(\Pi_{restname}(Rest) \cup Rest_2)$. Intuitively, Q combines restaurant rankings, using duplicate consolidation to collapse two occurrences of the same name to a single tuple. The only possible world of Q is (Tsukizi, Gagnaire, TourArgent), since duplicate elimination fails in the other possible worlds: indeed, this is the only possible way to combine the rankings.

We next show that the result of dupElim can still be represented as a po-relation, up to complete failure (which may be efficiently identified).

▶ **Theorem 41.** *For any po-relation* Γ*, we can test in PTIME if* dupElim(Γ) *completely fails; if it does not, we can compute in PTIME a po-relation* Γ' *such that* $pw(\Gamma') = \text{dupElim}(\Gamma)$.

We note that dupElim is not redundant with any of the other PosRA operators, generalizing Theorem 1:

▶ **Theorem 42.** *No operator among those of PosRA and* dupElim *can be expressed through a combination of the others.*

Last, we observe that dupElim can indeed be used to undo some of the effects of bag semantics. For instance, we can show the following:

▶ **Proposition 43.** *For any po-relation* Γ*, we have* $\text{dupElim}(\Gamma \cup \Gamma) = \text{dupElim}(\Gamma)$*: in particular, one completely fails iff the other does.*

We can also show that most of our previous tractability results still apply when the duplicate elimination operator is added:

▶ **Theorem 44.** *All* `POSS` *and* `CERT` *tractability results of Sections 4–6, except Theorem 23 and Theorem 31, extend to PosRA and* $\text{PosRA}^{\text{acc}}$ *where we allow* dupElim *(but impose the same restrictions on query operators, accumulation, and input po-relations).*

Furthermore, if in a set-semantics spirit we *require* that the query output has no duplicates, `POSS` and `CERT` are always tractable (as this avoids the technical difficulty of Example 11):

▶ **Theorem 45.** *For any PosRA query* Q*,* `POSS` *and* `CERT` *for* $\text{dupElim}(Q)$ *are in PTIME.*

Discussion. The introduced group-by and duplicate elimination operators have some shortcomings: the result of group-by is in general not representable by po-relations, and duplicate elimination may fail. These are both consequences of our design choices, where we capture only uncertainty on order (but not on tuple values) and design each operator so that its result corresponds to the result of applying it to each individual world of the input (see further discussion in Section 8). Avoiding these shortcomings is left for future work.

8 Comparison With Other Formalisms

We next compare our formalism to previously proposed formalisms: query languages over bags (with no order); a query language for partially ordered multisets; and other related work. To our knowledge, however, none of these works studied the possibility or certainty problems for partially ordered data, so that our technical results do not follow from them.

Standard bag semantics. We first compare to related work on the *bag semantics* for relational algebra. Indeed, a natural desideratum for our semantics on (partially) ordered relations is that it should be a faithful extension of bag semantics. We first consider the BALG^1 language on bags [21] (the "flat fragment" of their language BALG on nested relations). We denote by BALG^1_+ the fragment of BALG^1 that includes the standard extension of positive relational algebra operations to bags: additive union, cross product, selection, projection. We observe that, indeed, our semantics faithfully extends BALG^1_+: *query evaluation commutes with "forgetting" the order.* Formally, for a po-relation Γ, we denote by $\text{bag}(\Gamma)$ its underlying bag relation, and define likewise $\text{bag}(D)$ for a po-database D as the database of underlying bag relations. For the following comparison, we identify $\times_{\texttt{DIR}}$ and $\times_{\texttt{LEX}}$ with the $\times$ of [21] and our union with the additive union of [21]; the following holds:

▶ **Proposition 46.** *For any PosRA query* Q *and a po-relation* D*,* $\text{bag}(Q(D)) = Q(\text{bag}(D))$ *where* $Q(D)$ *is defined according to our semantics and* $Q(\text{bag}(D))$ *is defined by* $BALG^1_+$*.*

The full BALG^1 language includes additional operators, such as bag intersection and subtraction, which are non-monotone and as such may not be expressed in our language: it is also unclear how they could be extended to our setting (see further discussion in "Algebra on pomsets" below). On the other hand, BALG^1 does not include aggregation, and so $\text{PosRA}^{\text{acc}}$ and BALG^1 are incomparable in terms of expressive power.

A better yardstick to compare against for accumulation could be [33]: they show that their basic language $\mathcal{BQL}$ is equivalent to BALG, and then further extend the language

with aggregate operators, to define a language called $\mathcal{NRL}^{\text{aggr}}$ on nested relations. On flat relations, $\mathcal{NRL}^{\text{aggr}}$ captures functions that cannot be captured in our language: in particular the average function AVG is non-associative and thus cannot be captured by our accumulation function (which anyway focuses on order-dependent functions, as POSS/CERT are trivial otherwise). On the other hand, $\mathcal{NRL}^{\text{aggr}}$ cannot test parity (Corollary 5.7 in [33]) whereas this is easily captured by our accumulation operator. We conclude that $\mathcal{NRL}^{\text{aggr}}$ and PosRA$^{\text{acc}}$ are incomparable in terms of captured transformations on bags, even when restricted to flat relations.

Algebra on pomsets. We now compare our work to algebras defined on *pomsets* [20, 22], which also attempt to bridge partial order theory and data management (although, again, they do not study possibility and certainty). *Pomsets* are labeled posets quotiented by isomorphism (i.e., renaming of identifiers), like po-relations. A major conceptual difference between our formalism and that of [20, 22] is that their language focuses on processing *connected components* of the partial order graph, and their operators are tailored for that semantics. As a consequence, their semantics is *not* a faithful extension of bag semantics, i.e., their language would not satisfy the counterpart of Proposition 46 (see for instance the semantics of union in [20]). By contrast, we manipulate po-relations that stand for sets of possible list relations, and our operators are designed accordingly, unlike those of [20] where transformations take into account the structure (connected components) of the entire poset graph. Because of this choice, [20] introduces non-monotone operators that we cannot express, and can design a duplicate elimination operator that cannot fail. Indeed, the possible failure of our duplicate elimination operator is a direct consequence of its semantics of operating on each possible world, possibly leading to contradictions.

If we consequently disallow duplicate elimination in both languages for the sake of comparison, we note that the resulting fragment $\mathcal{P}$om-$\mathcal{A}$lg$_{\varepsilon_n}$ of the language of [20] can yield only series-parallel output (Proposition 4.1 of [20]), unlike PosRA queries whose output order may be arbitrary. Hence, $\mathcal{P}$om-$\mathcal{A}$lg$_{\varepsilon_n}$ does not subsume PosRA.

Incompleteness in databases. Our work is inspired by the field of incomplete information management, studied for various models [5, 30], in particular relational databases [24]. This field inspires our design of po-relations and study of possibility and certainty [3, 34]. However, uncertainty in these settings typically focuses on *whether* tuples exist or on their *values* (e.g., with nulls [10], including the novel approach of [31, 32]; with c-tables [24], probabilistic databases [42] or fuzzy numerical values as in [38]). To our knowledge, though, our work is the first to study possible and certain answers in the context of *order*-incomplete data. Combining order incompleteness with standard tuple-level uncertainty is left as a challenge for future work. Note that some works [8, 29, 32] use partial orders on *relations* to compare the informativeness of representations. This is unrelated to our partial orders on *tuples*.

Ordered domains. Another line of work has studied relational data management where the *domain elements* are (partially) ordered [25, 35, 43]. However, the perspective is different: we see order on tuples as part of the relations, and as being constructed by applying our operators; these works see order as being given *outside* of the query, hence do not study the propagation of uncertainty through queries. Also, queries in such works can often directly access the order relation [43, 6]. Some works also study uncertainty on totally ordered *numerical* domains [38, 39], while we look at general order relations.

Temporal databases. *Temporal databases* [9, 37] consider order on facts, but it is usually induced by timestamps, hence total. A notable exception is [16] which considers that some facts may be *more current* than others, with constraints leading to a partial order. In particular, they study the complexity of retrieving query answers that are certainly current, for a rich query class. In contrast, we can *manipulate* the order via queries, and we can also ask about aspects beyond currency, as shown throughout the paper (e.g., via accumulation).

Using preference information. Order theory has been also used to handle *preference information* in database systems [26, 4, 27, 2, 41], with some operators being the same as ours, and for *rank aggregation* [15, 26, 14], i.e. retrieving top-k query answers given multiple rankings. However, such works typically try to *resolve* uncertainty by reconciling many conflicting representations (e.g. via knowledge on the individual scores given by different sources and a function to aggregate them [15], or a preference function [2]). In contrast, we focus on maintaining a faithful model of *all* possible worlds without reconciling them, studying possible and certain answers in this respect.

9 Conclusion

This paper introduced an algebra for order-incomplete data. We have studied the complexity of possible and certain answers for this algebra, have shown the problems to be generally intractable, and identified multiple tractable cases. In future work we plan to study the incorporation of additional operators (in particular non-monotone ones), investigate how to combine order-uncertainty with uncertainty on values, and study additional semantics for dupElim. Last, it would be interesting to establish a dichotomy result for the complexity of `POSS`, and a complete syntactic characterization of cases where `POSS` is tractable.

Acknowledgements We are grateful to Marzio De Biasi, Pálvölgyi Dömötör, and Mikhail Rudoy, from `http://cstheory.stackexchange.com`, for helpful suggestions.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, 1995.

2 Bogdan Alexe, Mary Roth, and Wang-Chiew Tan. Preference-aware integration of temporal data. *PVLDB*, 8(4), 2014. `doi:10.14778/2735496.2735500`.

3 Lyublena Antova, Christoph Koch, and Dan Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *ICDT*, volume 4353 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2007. URL: `https://arxiv.org/abs/0705.4442`.

4 Anastasios Arvanitis and Georgia Koutrika. PrefDB: Supporting preferences as first-class citizens in relational databases. *IEEE TKDE*, 26(6), 2014.

5 Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with incomplete information. *J. ACM*, 58(1), 2010. `doi:10.1145/1870103.1870107`.

6 Michael Benedikt and Luc Segoufin. Towards a characterization of order-invariant queries over tame graphs. *Journal of Symbolic Logic*, 74, 2009.

7 Andeas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. Posets. In *Graph Classes. A Survey*, chapter 6. SIAM, 1987.

8 Peter Buneman, Achim Jung, and Atsushi Ohori. Using powerdomains to generalize relational databases. *TCS*, 91(1), 1991.

9 Jan Chomicki and David Toman. Time in database systems. In *Handbook of Temporal Reasoning in Artificial Intelligence*. Elsevier, 2005.

10 Edgar F. Codd. Extending the database relational model to capture more meaning. *TODS*, 4(4), 1979.

11 Latha S. Colby, Edward L. Robertson, Lawrence V. Saxton, and Dirk Van Gucht. A query language for list-based complex objects. In *PODS*, 1994.

12 Latha S. Colby, Lawrence V. Saxton, and Dirk Van Gucht. Concepts for modeling and querying list-structured data. *Information Processing & Management*, 30(5), 1994.

13 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 1950.

14 Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. Rank aggregation methods for the Web. In *WWW*, 2001.

15 Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

16 Wenfei Fan, Floris Geerts, and Jef Wijsen. Determining the currency of data. *TODS*, 37(4), 2012.

17 Roland Fraïssé. L'intervalle en théorie des relations; ses généralisations, filtre intervallaire et clôture d'une relation. *North-Holland Math. Stud.*, 99, 1984.

18 D. R. Fulkerson. Note on Dilworth's decomposition theorem for partially ordered sets. In *Proc. Amer. Math. Soc*, 1955.

19 Michael R. Garey and David S. Johnson. *Computers And Intractability. A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.

20 Stéphane Grumbach and Tova Milo. An algebra for pomsets. In *ICDT*, 1995.

21 Stéphane Grumbach and Tova Milo. Towards tractable algebras for bags. *JCSS*, 52(3), 1996. doi:10.1006/jcss.1996.0042.

22 Stéphane Grumbach and Tova Milo. An algebra for pomsets. *Inf. Comput.*, 150(2), 1999. doi:10.1006/inco.1998.2777.

23 John M. Howie. *Fundamentals of semigroup theory.* Oxford: Clarendon Press, 1995.

24 Tomasz Imieliński and Witold Lipski. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

25 Neil Immerman. Relational queries computable in polynomial time. *Inf. Control*, 68(1-3), 1986.

26 Marie Jacob, Benny Kimelfeld, and Julia Stoyanovich. A system for management and analysis of preference data. *VLDB Endow.*, 7(12), 2014.

27 Werner Kiessling. Foundations of preferences in database systems. In *VLDB*, 2002.

28 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002. doi:10.1145/543613.543644.

29 Leonid Libkin. A semantics-based approach to design of query languages for partial information. In *Semantics in Databases*, 1998.

30 Leonid Libkin. Data exchange and incomplete information. In *PODS*, 2006. doi:10.1145/1142351.1142360.

31 Leonid Libkin. Incomplete data: What went wrong, and how to fix it. In *PODS*, 2014.

32 Leonid Libkin. SQL's three-valued logic and certain answers. In *ICDT*, 2015. doi:10.4230/LIPIcs.ICDT.2015.94.

33 Leonid Libkin and Limsoon Wong. Query languages for bags and aggregate functions. *J. Comput. Syst. Sci.*, 55(2), 1997. doi:10.1006/jcss.1997.1523.

34 Witold Lipski, Jr. On semantic issues connected with incomplete information databases. *TODS*, 4(3), 1979.

35 Wilfred Ng. An extension of the relational data model to incorporate ordered domains. *TODS*, 26(3), 2001.

36 Bernd Schröder. *Ordered Sets: An Introduction*. Birkhäuser, 2003.

37 Richard T. Snodgrass, Jim Gray, and Jim Melton. *Developing time-oriented database applications in SQL*. Morgan Kaufmann, 2000.

38 Mohamed A. Soliman and Ihab F. Ilyas. Ranking with uncertain scores. In *ICDE*, 2009. doi:10.1109/ICDE.2009.102.

39 Mohamed A. Soliman, Ihab F. Ilyas, and Shalev Ben-David. Supporting ranking queries on uncertain and incomplete data. *VLDBJ*, 19(4), 2010.

40 Richard P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 1986.

41 Kostas Stefanidis, Georgia Koutrika, and Evaggelia Pitoura. A survey on representation, composition and application of preferences in database systems. *TODS*, 36(3), 2011.

42 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

43 Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *JCSS*, 54(1), 1997.

44 Manfred K Warmuth and David Haussler. On the complexity of iterated shuffle. *JCSS*, 28(3), 1984.

A Proof Sketches for Section 4 (General Complexity Results)

▶ **Theorem 12.** *The* POSS *problem is in NP for any PosRA or PosRA*$^{\text{acc}}$ *query. Further, there exists a PosRA query and a PosRA*$^{\text{acc}}$ *query for which the* POSS *problem is NP-complete.*

Proof Sketch. The membership for PosRA in NP is clear: guess a linear extension and check that it realizes the candidate possible result. For hardness, as in previous work [44], we reduce from the UNARY-3-PARTITION problem [19]: given a number B and $3m$ numbers written in unary, decide if they can be partitioned in triples that all sum to B. We reduce this to POSS for the identity PosRA query, on an arity-1 input po-relation where each input number n is represented as a chain of $n+2$ elements. The first and last elements of each chain are respectively called start and end markers, and elements of distinct chains are pairwise incomparable. The candidate possible world L consists of m repetitions of the following pattern: three start markers, B elements, three end markers. A linear extension achieves L iff the triples matched by $<$ to each copy of the pattern are a solution to UNARY-3-PARTITION, hence POSS for Q is NP-hard. This implies hardness for PosRA$^{\text{acc}}$, when accumulating with the identity map and concatenation (so that any list relation is mapped to itself). ◀

▶ **Theorem 13.** *The* CERT *problem is in coNP for any PosRA*$^{\text{acc}}$ *query, and there is a PosRA*$^{\text{acc}}$ *query for which it is coNP-complete.*

Proof Sketch. Again, membership is immediate. We show hardness of CERT by studying a PosRA$^{\text{acc}}$ query Q_{a} that checks if two input po-relations Γ and Γ' have some common possible world: Q_{a} does so so by testing if one can alternate between elements of Γ and Γ' with the same label, using accumulation in the transition monoid of a deterministic finite automaton. We show hardness of POSS for Q_{a} (as in the previous result), and further ensure that Q_{a} always has at most two possible accumulation results, no matter the input. Hence, POSS for Q_{a} reduces to the negation of CERT for Q_{a}, so that CERT is also hard. ◀

B Proof Sketches for Section 5 (Tractable Cases for POSS on PosRA Queries)

▶ **Theorem 17.** *The* POSS *problem is in NP for any PosRA or PosRA*$^{\text{acc}}$ *query. Further, there exists a PosRA query and a PosRA*$^{\text{acc}}$ *query for which the* POSS *problem is NP-complete.*

Proof Sketch. As $\times_{\texttt{DIR}}$ is disallowed, we can show that the po-relation $\Gamma := Q(D)$ has width k' depending only on k and the query Q (but not on D). We can then compute in PTIME a *chain partition* of Γ [13, 18], namely, a decomposition of Γ in totally ordered chains, with additional order constraints between them. This allows us to apply a dynamic algorithm to decide POSS: the state of the algorithm is the position on the chains. The number of states is polynomial with degree k', which is a constant when Q and k are fixed. ◀

▶ **Theorem 22.** *For any fixed $k \in \mathbb{N}$ and fixed PosRA query Q, the POSS problem for Q is in PTIME when all po-relations of the input po-database have ia-width $\leqslant k$.*

Proof Sketch. As in the proof of Theorem 17, we first show that the query result Γ also has ia-width depending only on k and the query. We then consider the order relation on indistinguishable antichains of Γ. For each linear extension τ of this order, we apply a greedy algorithm to decide POSS, for which we show correctness. The algorithm reads the candidate possible world in order and maps each tuple to an identifier of Γ with the correct value that was not mapped yet: we pick it in the first possible class according to the order τ. ◀

C Proof sketches for Section 6 (Tractable Cases for Accumulation Queries)

▶ **Theorem 26.** *CERT is in PTIME for any PosRA$^{\text{acc}}$ query that performs accumulation in a cancellative monoid.*

Proof Sketch. We show that the accumulation result in cancellative monoids is certain iff the po-relation on which we apply accumulation respects the following *safe swaps* criterion: for all tuples t_1 and t_2 and consecutive positions p and $p+1$ where they may appear, we have $h(t_1, p) \oplus h(t_2, p+1) = h(t_2, p) \oplus h(t_1, p+1)$. We can check this in PTIME. ◀