

Computing Possible and Certain Answers over Order-Incomplete Data

Antoine Amarilli, Mouhamadou Lamine Ba, Daniel Deutch, Pierre Senellart

▶ **To cite this version:**

Antoine Amarilli, Mouhamadou Lamine Ba, Daniel Deutch, Pierre Senellart. Computing Possible and Certain Answers over Order-Incomplete Data. Theoretical Computer Science, 2019, 797, pp.42-76. 10.1016/j.tcs.2019.05.013 . hal-01891814

HAL Id: hal-01891814
https://inria.hal.science/hal-01891814

Submitted on 20 Jul 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

Version of Record: https://www.sciencedirect.com/science/article/pii/S0304397519303214
Manuscript_5a5346aae94ebd477ffa2198611c185f

Computing Possible and Certain Answers over Order-Incomplete Data

Antoine Amarilli[a], Mouhamadou Lamine Ba[b], Daniel Deutch[c], Pierre Senellart[a,d,e]

[a] *LTCI, Télécom ParisTech, Université Paris-Saclay; Paris, France*
[b] *Université Alioune Diop de Bambey; Bambey, Senegal*
[c] *Blavatnik School of Computer Science, Tel Aviv University; Tel Aviv, Israel*
[d] *DI ENS, ENS, CNRS, PSL University; Paris, France*
[e] *Inria; Paris, France*

Abstract

This paper studies the complexity of query evaluation for databases whose relations are partially ordered; the problem commonly arises when combining or transforming ordered data from multiple sources. We focus on queries in a useful fragment of SQL, namely positive relational algebra with aggregates, whose bag semantics we extend to the partially ordered setting. Our semantics leads to the study of two main computational problems: the possibility and certainty of query answers. We show that these problems are respectively NP-complete and coNP-complete, but identify tractable cases depending on the query operators or input partial orders. We further introduce a duplicate elimination operator and study its effect on the complexity results.

Keywords: certain answer, possible answer, partial order, uncertain data

1. Introduction

Many applications need to combine and transform ordered data from multiple sources. Examples include sequences of readings from multiple sensors, or log entries from different applications or machines, that need to be combined to form a complete picture of events; rankings of restaurants and hotels based on various criteria (relevance, preference, or customer ratings); and concurrent edits of shared documents, where the order of contributions made by different users needs to be merged. Even if the order of items from each individual source is usually known, the order of items across sources is often *uncertain*. For instance, even when sensor readings or log entries are provided with timestamps, these may be ill-synchronized across sensors or machines; rankings of hotels and restaurants may be biased by different preferences of different users; concurrent contributions to documents may be ordered in multiple reasonable ways. We say that the resulting information is *order-incomplete*.

This paper studies query evaluation over order-incomplete data in a relational setting [1]. We focus on the running example of restaurants and hotels from a

© 2019 published by Elsevier. This manuscript is made available under the CC BY NC user license
https://creativecommons.org/licenses/by-nc/4.0/

travel website, ranked according to a proprietary function. An example query would ask for the ordered list of restaurant–hotel pairs such that the restaurant and hotel are in the same district, or such that the restaurant features a particular cuisine, and may further apply order-dependent operators to the result, e.g., limiting the output to the top-k such pairs, or aggregating a relevance score. To evaluate such queries, the initial order on the hotels and restaurants must be *preserved* through transformations. Furthermore, as we do not know how the proprietary order is defined, the result of transformations may become *uncertain*; hence, we need to represent all *possible* results that can be obtained depending on the underlying order.

Our approach is to handle this uncertainty through the classical notions of *possible and certain answers*. We say that there is a *certain answer* to the query when there is only one possible order on query results, or only one accumulation result, which is obtained no matter the order on the input and in intermediate results. In this case, it is useful to compute the certain answer, so that the user can then browse through the ordered query results (as is typically done when there is no 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*.

Our main contributions may be summarized as follows.

Model and Problem Definition (Sections 2, 3). Our work focuses on *bag semantics*, where a tuple may appear multiple times. Note that in the context of (partially) ordered relations, this means that multiple copies of the same tuple may appear in different "positions" in the order. For example, if we integrate multiple rankings of restaurants, then the same restaurant appears multiple times in different positions. We capture this model by a notion of *po-relations* (partially ordered) relations. A po-relation is essentially a relation accompanied with a partial order over its tuples; a technical subtlety is that each tuple is associated with an identifier (presumably internal and automatically generated), so that we have a way of referring to each tuple occurrence in the partial order (see further discussion in the problem definition below).

We then introduce a query language for partially ordered data. Our language design is guided by the goal of supporting SQL evaluation in presence of such data, and as such we focus on defining a semantics for an important fragment of SQL – namely positive relational algebra with aggregates. The semantics is "faithful" to SQL in the sense that, if we ignore order, then we get the standard SQL semantics. The notion corresponding to aggregation in our context is

LISP-like accumulation, whose semantics we extend to account for *partial* orders.

We view partially ordered relations as a concise representation of a set of possible worlds, namely, the linear extensions of the partial orders over the *underlying tuples of the relation*. For example, a linear extension is a ranked list of restaurant cuisines, where a cuisine may appear multiple times in the list. Note that in each such linear extension, the tuples appear *without* their respective internal identifiers which, as mentioned earlier, were only present in the po-relation as a technical tool.

Our definitions lead to a possible worlds semantics for query evaluation, and to two natural problems: whether a candidate answer – i.e., a ranked list of tuples (restaurants, cuisines, etc.) – is *possible*, i.e., is obtained for some possible world, and whether it is *certain*, i.e., is obtained for every possible world. Here again, note that in a candidate answer a tuple may appear multiple times, and naturally it appears without identifiers (which, as mentioned above, are internal and are unknown to the user). We formally define these two problems for our settings, and then embark on a study of their complexity.

Complexity in the General Case (Section 4). We first study the possibility and certainty problems without any restrictions on the input database. As usual in data management, given that queries are typically much smaller than databases, we study the *data complexity* of the problems, i.e., the complexity when the query is fixed. For our general definition of po-relations, we show that deciding whether an answer is possible is NP-complete, even without accumulation, and even for some very simple queries and input relations. In a particular case where we assume no duplicates – i.e., where tuples are uniquely identified, which means we are essentially back to the set semantics – possibility of an answer is in PTIME without accumulation, but is again NP-complete with accumulation. As for certainty, the problem can be decided in polynomial time in the case with no accumulation, but it is coNP-complete for queries with accumulation (even if we assume no duplicates in the input). Faced by the general intractability of the possibility and certainty problems, in the rest of the paper we search for restricted cases for which tractability holds.

Tractable Cases for Possibility Without Accumulation (Section 5). Even though possibility is NP-hard even without accumulation, we identify realistic cases where it is in fact tractable. In particular, we show that if the input relations are totally ordered then possibility is in PTIME for queries using a subset of our operators (all except the *direct product*). Assuming more severe restrictions on the query language, we further show tractability when some of the relations are (almost) ordered and the rest are (almost) unordered, as formalized via a newly introduced notion of *ia-width*.

Tractable Cases with Accumulation (Section 6). With accumulation, the certainty problem becomes intractable as well. Yet we show that if accumulation is captured by a finite cancellative monoid (in particular, if it is performed in a finite group), then certainty can again be decided in polynomial time. Further, we revisit the

tractability results for possibility from Section 5 and show that they extend to queries with accumulation under certain restrictions on the accumulation function.

Language Extensions (Section 7). We then study two extensions to our language, which are the counterparts of common SQL operators. The first is *group-by*, which allows us to group tuples for accumulation (as is done for aggregation in SQL with `GROUP BY`); we revisit our complexity results in its presence. The second is *duplicate elimination*: keeping a single representative of identical tuples, as in SQL with `SELECT DISTINCT`. In presence of order, it is challenging to design a semantics for this operator, and we discuss both semantic and complexity issues that arise from different possible definitions.

We compare our model and results to related work in Section 8, and conclude in Section 9.

This article is an extended version of the conference paper [2]. In contrast with the conference paper [2], all proofs are included here. We also discovered a bug in the proof of Theorem 22 of that paper [2], that also impacts Theorems 19 and 30 of [2]. Consequently, these results are omitted in the present paper.

2. Data Model and Query Language

We denote by $\mathbb{N}$ the set of nonnegative natural numbers and by $\mathbb{N}_{>0}$ the set of positive natural numbers, i.e., $\mathbb{N}_{>0} := \mathbb{N} \setminus \{0\}$. 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 $\langle v_1, \ldots, v_{\mathrm{a}(t)} \rangle$: for $1 \leqslant i \leqslant \mathrm{a}(t)$, we write $t.i$ to refer to v_i. The simplest notion of ordered relations are then *list relations* [3, 4]: 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 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 the Cartesian product, e.g., we can use tuples of natural numbers), a *strict partial order* $<$ on ID, and a (generally non-injective) mapping T from ID to $\mathcal{D}^n$. The *domain* of Γ is the subset of values of $\mathcal{D}$ that occur in the image of T. The actual identifiers in ID 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 denote them (ID, T). Another special case is that of *totally ordered po-relations*, where $<$ is a total order.

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 $< \subseteq <'$, i.e., 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. Note that each such linear extension "strips away" the identifiers and includes only the tuples. For instance, if Γ is unordered, then $pw(\Gamma)$ consists of all permutations of the tuples of Γ; and if Γ is totally ordered then $pw(\Gamma)$ contains exactly one possible world.

Po-relations can thus model uncertainty over the *order* of tuples. However, note that they cannot model uncertainty on tuple *values*. Specifically, let us define the *underlying bag relation* of a po-relation $\Gamma = (ID, T, <)$ as (ID, T). Unlike order, this underlying bag relation is always certain.

We extend some classical notions from partial order theory to po-relations.

Letting $\Gamma = (ID, T, <)$ be a po-relation, an *order ideal* of Γ is a subset $S \subseteq ID$ such that, for all $x, y \in ID$, if $x < y$ and $y \in S$ then $x \in S$. An *antichain* [5] of Γ is a set $A \subseteq ID$ of pairwise incomparable tuple identifiers. The *width* of Γ is the size of its largest antichain, and 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 number of tuples; the width of a po-relation can be computed in polynomial time [6].

A *chain partition* of Γ is a partition $ID = \Lambda_1 \sqcup \cdots \sqcup \Lambda_n$ such that the restriction of $<$ to each Λ_i is a total order: we call each Λ_i a *chain*. Note that $<$ may include comparability relations across chains, i.e., relating elements in Λ_i to elements in Λ_j for $i \neq j$. The *width* of the chain partition is n. By Dilworth's theorem [7, 6], the width w of Γ is the smallest possible width of a chain partition of Γ; furthermore, given Γ, we can compute in polynomial time both its width w and a chain partition of Γ of width w.

2.1. PosRA: Queries Without Accumulation

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 with *accumulation* in Section 2.3, and add further operations 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 involve constant values in $\mathcal{D}$ and tuple attributes written as $.i$ for $i \in \mathbb{N}_{>0}$. For instance, the selection $\sigma_{.1 \neq \text{"a"} \wedge .2 \neq .3}$ selects tuples whose first attribute is equal to the constant "a" and whose second attribute is different from their third attribute.

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 $\Pi_{A_1,\ldots,A_n}(T(id)) := \langle T(id).A_1, \ldots, T(id).A_n\rangle$.

As for *union*, we impose the minimal order constraints that are compatible with those of the inputs. We use the *parallel composition* [8] of two partial orders $<$ and $<'$ on disjoint sets ID and ID', i.e., the partial order $<'' := (< \cup <')$ on $ID \cup ID'$. Note that $<''$ is the same order as $<$ on ID and as $<'$ on ID', and that all elements from ID are incomparable to all elements from ID'.

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 $\Gamma \cup \Gamma' := (ID \cup ID', T'', (< \cup <'))$, 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 how we renamed 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, as in [9], the *direct product* $<_{\texttt{DIR}} := (< \times_{\texttt{DIR}} <')$ of two partial orders $<$ and $<'$ on sets ID and ID' is defined by $(id_1, id_1') <_{\texttt{DIR}} (id_2, id_2')$ iff $id_1 < id_2$ and $id_1' <' id_2'$ for each $(id_1, id_1'), (id_2, id_2') \in ID \times ID'$. 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 set 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* [9]) $<_{\texttt{LEX}} := (< \times_{\texttt{LEX}} <')$ of two partial orders $<$ and $<'$, defined by $(id_1, id_1') <_{\texttt{LEX}} (id_2, id_2')$ iff either $id_1 < id_2$, or $id_1 = id_2$ and $id_1' <' id_2'$, for all $(id_1, id_1'), (id_2, id_2') \in ID \times ID'$.

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 the direct product.

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

restname	*district*
Gagnaire	8
TourArgent	5

(a) *Restaurant* table

hotelname	*district*
Mercure	5
Balzac	8
Mercure	12

(b) *Hotel* table

hotelname	*district*
Balzac	8
Mercure	5
Mercure	12

(c) $Hotel_2$ table

Figure 1: Running example: Paris restaurants and hotels

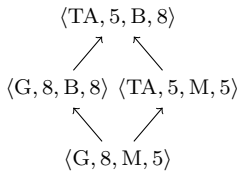


Figure 2: Example 1

constant expressions:

- 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 $pw([\leqslant n]) = \{(1, \ldots, n)\}$.

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 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 $|Q|$ the number of symbols of Q, and we denote by $Q(D)$ the po-relation obtained by evaluating Q over D.

Example 1. *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. For brevity, we do not represent identifiers in po-relations, and we also deviate slightly from our formalism by adopting the named perspective in examples, i.e., giving names to attributes.*

Let $Q := Restaurant \times_{DIR} (\sigma_{district \neq \text{“12”}}(Hotel))$. Its result $Q(D)$ has two possible worlds, where we abbreviate hotel and restaurant names:

- $(\langle \mathrm{G}, 8, \mathrm{M}, 5\rangle, \langle \mathrm{G}, 8, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, 5, \mathrm{M}, 5\rangle, \langle \mathrm{TA}, 5, \mathrm{B}, 8\rangle)$*;*
- $(\langle \mathrm{G}, 8, \mathrm{M}, 5\rangle, \langle \mathrm{TA}, 5, \mathrm{M}, 5\rangle, \langle \mathrm{G}, 8, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, 5, \mathrm{B}, 8\rangle)$.

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, again writing tuple values instead of tuple identifiers for brevity: note that, following the usual convention for Hasse diagrams in partial order theory, the order in Figure 2 is drawn in the reverse direction of that of Figure 1, i.e., from bottom to top.*

Consider now the query $Q' := \Pi(\sigma_{Restaurant.district = Hotel.district}(Q))$, where Π projects out Hotel.district. 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 $\times_{DIR}$ by $\times_{LEX}$ in Q, we have $pw(Q''(D)) = (\langle \mathrm{G}, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, \mathrm{M}, 5\rangle)$.

It is easy to show that we can efficiently evaluate PosRA queries on po-relations, which we will use throughout the sequel.

Proposition 2. *For any fixed PosRA query Q, given a po-database D, we can construct the po-relation $Q(D)$ in time $O\big(|D|^{|Q|}\big)$, i.e., in polynomial time in the size of D.*

Proof. We show the claim by a simple induction on the query Q, noting that $|Q|$ is at least $k+1$, where k is the number of operators in Q.

- If Q is a relation name R, then $Q(D)$ is obtained in linear time.
- If Q is a constant expression, then $Q(D)$ is obtained in constant time.
- If $Q = \sigma_\psi(Q')$ or $Q = \Pi_{k_1 \dots k_p}(Q')$, then $Q(D)$ is obtained in time linear in $|Q'(D)|$, and we conclude by the induction hypothesis.
- If $Q = Q_1 \cup Q_2$ or $Q = Q_1 \times_{\texttt{LEX}} Q_2$ or $Q = Q_1 \times_{\texttt{DIR}} Q_2$, then $Q(D)$ is obtained in time linear in $|Q_1(D)| \times |Q_2(D)|$, and we conclude again by the induction hypothesis. □

Note that Proposition 2 computes the result of a query as a po-relation Γ. However, we cannot efficiently compute the complete set $pw(\Gamma)$ of possible worlds of Γ, even if all relations of the input po-database are totally ordered. For instance, consider the query $Q := R \cup S$, and a po-database D interpreting R and S as totally ordered relations with disjoint domains and with n tuples each. It is easy to see that the query result $Q(D)$ has $\binom{2n}{n}$ possible worlds, which is exponential in D. This intractability is the reason why will we study the possibility and certainty problems in the sequel.

2.2. Incomparability of PosRA Operators

Before extending our query language with accumulation, we address the natural question of whether any of our operators is subsumed by the others. We show that this is not the case.

Theorem 3. *No PosRA operator can be expressed through a combination of the others.*

We prove Theorem 3 in the rest of this subsection. We consider each operator in turn, and show that it cannot be expressed through a combination of the others. We first consider constant expressions and show differences in expressiveness even when setting the input po-database to be empty.

- For $[t]$, consider the query $[\langle 0 \rangle]$. The value 0 is not in the database, and cannot be produced by the $[\leqslant n]$ constant expression, and so this query has no equivalent that does not use the $[t]$ constant expression.
- For $[\leqslant n]$, observe that $[\leqslant 2]$ is a po-relation with a non-empty order, while any query involving the other operators will have empty order (none of our unary and binary operators turns unordered po-relations into an ordered one, and the $[t]$ constant expression produces an unordered po-relation).

Moving on to unary and binary operators, all operators but products are easily shown to be non-expressible.

selection. For any constant a not in $\mathbb{N}$, consider the po-database D_a consisting of a single unordered po-relation with name R formed of two unary tuples $\langle 0 \rangle$ and $\langle a \rangle$. Let $Q = \sigma_{.1 \neq \text{“0”}}(R)$. Then, $Q(D_a)$ is the po-relation consisting only of the tuple $\langle a \rangle$. No PosRA query without selection has the same semantics, as no other operator than selection can create a po-relation containing the constant a for any input D_a, unless it also contains the constant 0.

projection. Π is the only operator that can decrease the arity of an input po-relation.

union. $[\langle 0 \rangle] \cup [\langle 1 \rangle]$ (over the empty po-database) cannot be simulated by any combination of operators, as can be simply shown by induction: no other operator will produce a po-relation which has the two elements 0 and 1 in the same attribute.

Observe that product operators are the only ones that can increase arity, so taken together they are non-redundant with the other operators. Hence, it only remains to show that each of $\times_{\texttt{DIR}}$ and $\times_{\texttt{LEX}}$ is not redundant. To do this, let us call $\text{PosRA}_{\texttt{LEX}}$ the fragment of PosRA that disallows the $\times_{\texttt{DIR}}$ operator, but allows all other operators (including $\times_{\texttt{LEX}}$). We also define $\text{PosRA}_{\texttt{DIR}}$ that disallows $\times_{\texttt{LEX}}$ but not $\times_{\texttt{DIR}}$.

We will first show that the $\times_{\texttt{DIR}}$ product is not redundant, which we will do using the notion of width. Specifically, consider the query $Q_{\texttt{DIR}} = R \times_{\texttt{DIR}} R$ and an input po-database D_n where R is mapped to $[\leqslant n]$ (an input relation of width 1) for an arbitrary R_n. It is then clear that the po-relation $Q(D_n)$ has width n. We will show that this query cannot be captured in $\text{PosRA}_{\texttt{LEX}}$, because $\text{PosRA}_{\texttt{LEX}}$ queries can only make width increase in a way that depends on the *width* of the input po-relations, but not on their *size*.

Lemma 4. *Let $k \geqslant 2$ and Q be a $\text{PosRA}_{\texttt{LEX}}$ query. For any po-database D of width $\leqslant k$, the po-relation $Q(D)$ has width $\leqslant k^{|Q|+1}$.*

Proof. We first show by induction on the $\text{PosRA}_{\texttt{LEX}}$ query Q that the width of the query output can be bounded as a function of k. For the base case, the input po-relations have width $\leqslant k$, and all constant po-relations (singletons and constant chains) have width 1. Let us show the induction step.

- Given two po-relations Γ_1 and Γ_2 with width respectively k_1 and k_2, their union $\Gamma := \Gamma_1 \cup \Gamma_2$ clearly has width at most $k_1 + k_2$. Indeed, any antichain in Γ must be the union of an antichain of Γ_1 and of an antichain of Γ_2.
- Given a po-relation Γ_1 with width k_1, applying a projection or selection to Γ_1 cannot increase the width.

- Given two po-relations $\Gamma_1 = (ID_1, T_1, <_1)$ and $\Gamma_2 = (ID_2, T_2, <_2)$ with width respectively k_1 and k_2, their product $\Gamma := \Gamma_1 \times_{\texttt{LEX}} \Gamma_2$ has width at most $k_1 \cdot k_2$. To show this, write $\Gamma = (ID, T, <)$, consider any set $A \subseteq ID$ of cardinality $> k_1 \cdot k_2$, and let us argue that A is not an antichain. By the definition of $\times_{\texttt{LEX}}$, we can see each identifier of A as an element of $ID_1 \times ID_2$. Now, one of the following must hold.

  1. Letting S_1 be the set of identifiers $u \in ID_1$ for which we have $(u, v) \in A$ for some $v \in ID_2$, it is the case that $|S_1| > k_1$.
  2. There exists u such that, letting $S_2(u) := \{v \mid (u, v) \in A\}$, we have $|S_2(u)| > k_2$.

  Informally, when putting $> k_1 \cdot k_2$ values in buckets (the value of their first component), either $> k_1$ different buckets are used, or there is a bucket containing $> k_2$ elements.

  In the first case, as $S_1 \subseteq ID_1$, as $|S_1| > k_1$, and as Γ_1 has width k_1, we know that S_1 cannot be an antichain, so it must contain two comparable elements $u <_1 u'$. Hence, considering any $v, v' \in ID_2$ such that $w = (u, v)$ and $w' = (u', v')$ are in A, we have by the definition of $\times_{\texttt{LEX}}$ that $w < w'$, so that A is not an antichain. In the second case, as $S_2(u) \subseteq ID_2$, as $|S_2(u)| > k_2$, and as Γ_2 has width k_2, we know that $S_2(u)$ cannot be an antichain, so it must contain two comparable elements $v <_2 v'$. Hence, considering $w = (u, v)$ and $w' = (u, v')$ which are in A, we have $w < w'$, and again A is not an antichain. Hence, no set of cardinality $> k_1 \cdot k_2$ of Γ is an antichain, so Γ has width $\leqslant k_1 \cdot k_2$ as claimed.

Second, we explain why the bound on the width of the query output can be chosen as in the lemma statement. Specifically, letting o be the number of product operators in Q plus the number of union operators, we show that we can bound the width of $Q(D)$ by k^{o+1}. Indeed, the output of queries without product or union operators have width at most k (because $k \geqslant 1$). Further, as projections and selections do not change the width, the only operators to consider are product and union. For the union operator, if Q_1 has o_1 such operators and Q_2 has o_2 such operators, bounding inductively the width of $Q_1(D)$ by k^{o_1+1} and $Q_2(D)$ by k^{o_2+1}, for $Q := Q_1 \cup Q_2$, the number of union and product operators is $o_1 + o_2 + 1$, and the new bound is $k^{o_1+1} + k^{o_2+1}$, which is $\leqslant k^{o_1+1+o_2+1}$ because $k \geqslant 2$, i.e., it is $\leqslant k^{(o_1+o_2+1)+1}$. For the $\times_{\texttt{LEX}}$ operator, we proceed in the same way and directly obtain the $k^{(o_1+o_2+1)+1}$ bound. Hence, we can indeed bound the width of $Q(D)$ by $k^{|Q|+1}$ as given in the statement, which concludes the proof. □

We have shown Lemma 4: $\text{PosRA}_{\texttt{LEX}}$ queries can only make the width increase as a function of the query and of the width of the input po-relations. Hence, the query $Q_{\texttt{DIR}}$ cannot be captured in $\text{PosRA}_{\texttt{LEX}}$, and the $\times_{\texttt{DIR}}$ product is not redundant.

Conversely, let us show that the $\times_{\texttt{LEX}}$ product is not redundant. To do this, we introduce the *concatenation* of po-relations.

Definition 5. *The* concatenation $\Gamma \cup_{\mathit{CAT}} \Gamma'$ *of two po-relations* Γ *and* Γ' *is the series composition of their two partial orders. Note that* $pw(\Gamma \cup_{\mathit{CAT}} \Gamma') = \{L \cup_{\mathit{CAT}} L' \mid L \in pw(\Gamma), L' \in pw(\Gamma')\}$*, where* $L \cup_{\mathit{CAT}} L'$ *is the concatenation of two list relations in the usual sense.*

We show that concatenation can be captured in $\mathrm{PosRA}_{\mathtt{LEX}}$.

Lemma 6. *For any arity* $n \in \mathbb{N}$ *and distinguished relation names* R *and* R'*, there is a query* Q_n *without* $\times_{\mathit{DIR}}$ *operator such that, for any two po-relations* Γ *and* Γ' *of arity* n*, letting* D *be the database mapping* R *to* Γ *and* R' *to* Γ'*, the query result* $Q_n(D)$ *is* $\Gamma \cup_{\mathit{CAT}} \Gamma'$*.*

Proof. For any $n \in \mathbb{N}$ and names R and R', consider the following query:

$$Q_n(R, R') := \Pi_{3 \ldots n+2} \left(\sigma_{.1=.2} \left([\leqslant 2] \times_{\mathtt{LEX}} (([1] \times_{\mathtt{LEX}} R) \cup ([2] \times_{\mathtt{LEX}} R'))\right)\right)$$

It is easily verified that Q_n satisfies the claimed property. □

By contrast, we show that concatenation cannot be captured in $\mathrm{PosRA}_{\mathtt{DIR}}$.

Lemma 7. *For any arity* $n \in \mathbb{N}_{>0}$ *and distinguished relation names* R *and* R'*, there is no* $\mathit{PosRA}_{\mathit{DIR}}$ *query* Q_n *such that, for any po-relations* Γ *and* Γ' *of arity* n*, letting* D *be the po-database that maps* R *to* Γ *and* R' *to* Γ'*, the query result* $Q_n(D)$ *is* $\Gamma \cup_{\mathit{CAT}} \Gamma'$*.*

To prove Lemma 7, we first introduce the following concept.

Definition 8. *Let* $v \in \mathcal{D}$*. We call a po-relation* $\Gamma = (\mathit{ID}, T, <)$ v-impartial *if, for any two identifiers* id_1 *and* id_2 *and* $1 \leqslant i \leqslant \mathrm{a}(\Gamma)$ *such that exactly one of* $T(id_1).i$*,* $T(id_2).i$ *is* v*, the following holds:* id_1 *and* id_2 *are* incomparable*, namely, neither* $id_1 < id_2$ *nor* $id_2 < id_1$ *hold.*

Lemma 9. *Let* $v \in \mathcal{D} \backslash \mathbb{N}$ *be a value. For any* $\mathit{PosRA}_{\mathit{DIR}}$ *query* Q*, for any po-database* D *of* v*-impartial po-relations, the po-relation* $Q(D)$ *is* v*-impartial.*

Proof. Let D be a po-database of v-impartial po-relations. We show by induction on the query Q that v-impartiality is preserved. The base cases are the following.

- For the base relations, the claim is vacuous by our hypothesis on D.
- For the singleton constant expressions, the claim is trivial as they contain less than two tuples.
- For the $[\leqslant i]$ constant expressions, the claim is immediate as $v \notin \mathbb{N}$.

We now prove the induction step.

- For selection, the claim is shown by noticing that, for any v-impartial po-relation Γ, letting Γ' be the image of Γ by any selection, Γ' is itself v-impartial. Indeed, considering two identifiers id_1 and id_2 in Γ' and $1 \leqslant i \leqslant \mathrm{a}(\Gamma)$ satisfying the condition, as Γ is v-impartial, id_1 and id_2 are incomparable in Γ, so they are also incomparable in Γ'.

- For projection, the claim is also immediate as the property to prove is maintained when reordering, copying or deleting attributes. Indeed, considering again two identifiers id'_1 and id'_2 of Γ' and $1 \leqslant i' \leqslant \mathrm{a}(\Gamma')$, the respective preimages id_1 and id_2 in Γ of id'_1 and id'_2 satisfy the same condition for some different $1 \leqslant i \leqslant \mathrm{a}(\Gamma)$ which is the attribute in Γ that was projected to give attribute i' in Γ', so we again use the impartiality of the original po-relation to conclude.

- For union, letting $\Gamma'' := \Gamma \cup \Gamma'$, and writing $\Gamma'' = (ID'', T'', <'')$, assume by contradiction the existence of two identifiers $id_1, id_2 \in \Gamma''$ and $1 \leqslant i \leqslant \mathrm{a}(\Gamma'')$ such that exactly one of $T''(id_1).i$ and $T''(id_2).i$ is v but (without loss of generality) $id_1 < id_2$ in Γ''. It is easily seen that, as id_1 and id_2 are not incomparable, they must come from the same relation; but then, as that relation was v-impartial, we have a contradiction.

- For $\times_{\texttt{DIR}}$, consider $\Gamma'' := \Gamma \times_{\texttt{DIR}} \Gamma'$ where Γ and Γ' are v-impartial, and write $\Gamma'' = (ID'', T'', <'')$ as above. Assume that there are two identifiers id''_1 and id''_2 of ID'' and $1 \leqslant i \leqslant \mathrm{a}(\Gamma'')$ that violate the v-impartiality of Γ''. Let $(id_1, id'_1), (id_2, id'_2) \in ID \times ID'$ be the pairs of identifiers used to create id''_1 and id''_2. We distinguish on whether $1 \leqslant i \leqslant \mathrm{a}(\Gamma)$ or $\mathrm{a}(\Gamma) < i \leqslant \mathrm{a}(\Gamma)+\mathrm{a}(\Gamma')$. In the first case, we deduce that exactly one of $T(id_1).i$ and $T(id_2).i$ is v, so that in particular $id_1 \neq id_2$. Thus, by the definition of the order in $\times_{\texttt{DIR}}$, it is easily seen that, because id''_1 and id''_2 are comparable in Γ'', id_1 and id_2 must compare in the same way in Γ, contradicting the v-impartiality of Γ. The second case is symmetric. □

We now conclude with the proof of Lemma 7.

Proof. Let us assume by way of contradiction that there is $n \in \mathbb{N}_{>0}$ and a $\mathrm{PosRA}_{\texttt{DIR}}$ query Q_n that captures $\cup_{\texttt{CAT}}$. Let $v \neq v'$ be two distinct values in $\mathcal{D} \backslash \mathbb{N}$, and consider the singleton po-relation Γ containing one identifier of value t and Γ' containing one identifier of value t', where t (resp. t') are tuples of arity n containing n times the value v (resp. v'). Consider the po-database D mapping R to Γ and R' to Γ'. Write $\Gamma'' := Q_n(D)$. By our assumption, as $\Gamma'' = (ID'', T'', <'')$ is $\Gamma \cup_{\texttt{CAT}} \Gamma'$, it must contain an identifier $id \in ID''$ such that $T''(id) = t$ and an identifier $id' \in ID''$ such that $T''(id') = t'$. Now, as Γ and Γ' are (vacuously) v-impartial, Lemma 9 implies that Γ'' is v-impartial. Hence, as $n > 0$, taking $i = 1$, as $t \neq t'$ and exactly one of $t.1$ and $t'.1$ is v, the identifiers id and id' are incomparable in $<''$, so there is a possible world of Γ'' where id' precedes id. This contradicts the fact that, as we should have $\Gamma'' = \Gamma \cup_{\texttt{CAT}} \Gamma'$, the po-relation Γ'' should have exactly one possible world, namely, (t, t'). □

This establishes that the $\times_{\texttt{LEX}}$ operator cannot be expressed using the others, and shows that none of our operators is redundant, which concludes the proof of Theorem 3.

2.3. PosRA$^{\text{acc}}$: Queries With 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. Recall that a *monoid* $(\mathcal{M}, \oplus, \varepsilon)$ consists of a set $\mathcal{M}$ (not necessarily finite), an associative operation $\oplus : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$, and an element $\varepsilon \in \mathcal{M}$ which is neutral for $\oplus$, i.e., for all $m \in \mathcal{M}$, we have $\varepsilon \oplus m = m \oplus \varepsilon = m$. We will use a monoid as the structure in which we perform accumulation. We can now define accumulation on a given list relation.

Definition 10. *For $k \in \mathbb{N}$, let $h : \mathcal{D}^k \times \mathbb{N}_{>0} \to \mathcal{M}$ be a function called an* arity-k accumulation map*, which maps pairs consisting of an k-tuple and a position to a value in the monoid $\mathcal{M}$. We call* $\text{accum}_{h,\oplus}$ *an* arity-k accumulation operator*; its result* $\text{accum}_{h,\oplus}(L)$ *on an arity-k list relation $L = (t_1, \ldots, t_n)$ is $h(t_1, 1) \oplus \cdots \oplus h(t_n, n)$, and it is ε if L is empty. For complexity purposes, we* always *require accumulation operators to be* PTIME-evaluable*, i.e., we can evaluate the accumulation map and the monoid operator in time polynomial in their inputs, and we can compute* $\text{accum}_{h,\oplus}(L)$ *in polynomial time on any input list relation L.*

Intuitively, the accumulation operator maps each occurrence of a tuple in the list with h to $\mathcal{M}$, where accumulation is performed with $\oplus$. (Remember that the input L to the accumulation is a list relation, so each tuple occurrence has a specific position.) 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.

We will often look at special cases for accumulation, especially when deriving complexity results. Here are the restrictions that we will consider.

Definition 11. *We say that an accumulation operator is* position-invariant *if its accumulation map ignores the second input, so that effectively its only input is the tuple itself*

We say that an accumulation operator is finite *if its monoid $(\mathcal{M}, \oplus, \varepsilon)$ is finite.*

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 $b \oplus a = c \oplus a$ implies $b = c$. We call $\mathcal{M}$ a* cancellative monoid *[10] if all its elements are cancellable. We say that an accumulation operator is* cancellative *if its monoid is.*

Note that, in particular, a group is always cancellative, but there are some cancellative monoids which are not groups, e.g., the monoid of concatenation.

We can now define the language PosRA$^{\text{acc}}$ that contains all queries of the form $Q = \text{accum}_{h,\oplus}(Q')$, where $\text{accum}_{h,\oplus}$ 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 12. *For a po-relation Γ, we define* $\text{accum}_{h,\oplus}(\Gamma) := \{\text{accum}_{h,\oplus}(L) \mid L \in pw(\Gamma)\}$.

Of course, accumulation has exactly one result whenever the accumulation 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., the monoid $\mathcal{M}$ of strings with $\oplus$ being the concatenation operation. In this case, there may be more than one accumulation result.

Example 13. *As a first example, let* $Ratings(user, restaurant, rating)$ *be an* unordered *po-relation describing the numerical ratings given by users to restaurants, where each user rated each restaurant at most once. Let* $Relevance(user)$ *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* $Q_1 := \text{accum}_{h_1,+}(\sigma_\psi(Relevance \times_{LEX} Ratings))$ *where we set* $h_1(t, n) := t.rating \times w(n)$*, and where* ψ *is the tuple predicate:* $restaurant =$ *"Gagnaire"* $\wedge$ $Ratings.user = Relevance.user$*. The query* Q_1 *gives the total rating of "Gagnaire", and each possible world of Relevance may lead to a different accumulation result. This accumulation operator is cancellative, but it is neither position-invariant nor finite.*

As a second example, consider an unordered po-relation $HotelCity(hotel, city)$ *indicating in which city each hotel is located, and consider a po-relation* $City(city)$ *which is (partially) ranked by a criterion such as interest level, proximity, etc. Now consider the query* $Q_2 := \text{accum}_{h_2,\text{concat}}(\Pi_{hotel}(Q_2'))$*, with* $Q_2' := \sigma_{City.city=HotelCity.city}(City \times_{LEX} HotelCity)$ *and* $h_2(t, n) := t$*. Here, the operator* "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. This accumulation operator is cancellative and position-invariant, but it is not finite.*

3. Possibility and Certainty

Evaluating a PosRA or $\text{PosRA}^{\text{acc}}$ query Q on a po-database D yields a *set of possible results*: for $\text{PosRA}^{\text{acc}}$, 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 on the result may come from uncertainty on the order of the input relations (i.e., if they are po-relations with multiple possible worlds), but it may also be caused by the query, e.g., the union of two non-empty totally ordered relations is not totally ordered. In some cases, however, there is only one possible result to the query, i.e., a *certain* answer. In other cases, we may wish to examine multiple *possible* answers. We thus define the corresponding problems.

Definition 14 (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 of* Q *on* D. *The* certainty problem *(*`CERT`*) asks if* $pw(Q(D)) = \{L\}$, *i.e., if* L *is the only possible result of* Q *on* D.

Likewise, if Q *is a PosRA*$^{\text{acc}}$ *query with an accumulation monoid* $\mathcal{M}$, *for a result* $v \in \mathcal{M}$, *the* `POSS` *problem asks whether* $v \in Q(D)$, *and* `CERT` *asks whether* $Q(D) = \{v\}$.

For PosRA$^{\text{acc}}$, our definition follows the usual notion of possible and certain answers in data integration [11] and incomplete information [12]. For PosRA, we ask for possibility or certainty of an *entire* output list relation of tuples *without identifiers*: indeed, as we explained above, the identifiers are only internally generated and thus expected to be unknown to the user. These problems correspond to *instance possibility and certainty* [13]. 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 a complex query 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 it is the case, then we can compute the certain output in polynomial time, by choosing an arbitrary linear extension and computing the corresponding possible world. However, `CERT` is a challenging problem to solve, because of duplicate values (see the "Technical difficulties" paragraph below).

Example 15. *Consider the po-database* D *of Figure 1 with relations Restaurant and* $Hotel_2$. *To find recommended pairs of hotels and restaurants in the same district, we can write* $Q := \sigma_{Restaurant.district=Hotel_2.district}(Restaurant \times_{DIR} Hotel_2)$. *Evaluating* $Q(D)$ *yields the list relation* $(\langle G, 8, B, 8\rangle, \langle TA, 5, M, 5\rangle)$ *as a unique possible world: it is a* certain *result.*

We may also obtain a certain result in cases when the input relations are larger. 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, we can instead try to decide whether some list relations are a possible answer. This 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: this can be done by checking if the ranking on the pairs is a possible result of the query that unifies the hotel ranking and restaurant ranking.

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 13, 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 *position*[1] of particular tuples of interest.

One particular application of accumulation is to model *position-based selection* queries. Consider for instance a *top-k* operator, defined on list relations, which retrieves a list relation of the first k tuples. Let us extend the top-k operator to po-relations in the expected way: the set of top-k results on a po-relation Γ is the set of top-k results on the list relations of $pw(\Gamma)$. We can implement *top-k* as $\mathrm{accum}_{h_3,\mathrm{concat}}$ 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 $\mathrm{accum}_{h_4,\mathrm{concat}}$ with $h_4(t,n)$ being (t) for $n = k$ and ε otherwise. Both these accumulation operators are cancellative because they use the concatenation monoid, and they are finite if we assume that the domain of the output is fixed (e.g., ratings in $\{1,\dots,10\}$), and if we also assume for top-k that k is fixed.

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 $\oplus$ that returns its first argument: assuming that t_1 and t_2 are both present, the result is $\top$ if the first occurrence of t_1 precedes any occurrence of t_2, and it is $\bot$ otherwise. This accumulation operator is finite and position-invariant, but not cancellative.

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

Example 16. *Let* $Q := \Pi_{district}(\sigma_{Restaurant.district=Hotel.district}(Restaurant \times_{\texttt{DIR}} Hotel))$*, that computes ordered recommendations of districts including both hotels and restaurants. The user can use accumulation to compute the best district to stay in with* $Q' = top\text{-}1(Q)$*. When* Q' *has a certain answer, 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*$^{\mathrm{acc}}$ *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 though we can compute the po-relation $Q(D)$ in polynomial time (by Proposition 2) and present it to the user, they still cannot easily determine the possible and certain answers out of the po-relation.

[1] Remember that the *existence* of a tuple is not order-dependent, so it is trivial to check in our setting.

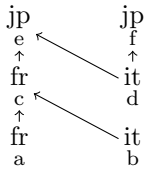


Figure 3: Po-relation in Example 17

Example 17. *Consider a po-relation* $\Gamma = (ID, T, <)$ *with* $ID = \{id_\mathrm{a}, id_\mathrm{b}, id_\mathrm{c}, id_\mathrm{d}, id_\mathrm{e}, id_\mathrm{f}\}$*, with* $T(id_\mathrm{a}) := \langle Gagnaire, fr\rangle$*,* $T(id_\mathrm{b}) := \langle Italia, it\rangle$*,* $T(id_\mathrm{c}) := \langle TourArgent, fr\rangle$*,* $T(id_\mathrm{d}) := \langle Verdi, it\rangle$*,* $T(id_\mathrm{e}) := \langle Tsukizi, jp\rangle$*,* $T(id_\mathrm{f}) := \langle Sola, jp\rangle$*, and with* $id_\mathrm{a} < id_\mathrm{c}$*,* $id_\mathrm{b} < id_\mathrm{c}$*,* $id_\mathrm{c} < id_\mathrm{e}$*,* $id_\mathrm{d} < id_\mathrm{e}$*, and* $id_\mathrm{d} < id_\mathrm{f}$*. 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* (it, fr, jp, it, fr, jp)*, and consider* `POSS` *for* Q*,* Γ*, and* L*.*

We have that $L \in pw(Q(\Gamma))$*, as shown by the linear extension* $id_\mathrm{d} <' id_\mathrm{a} <' id_\mathrm{f} <' id_\mathrm{b} <' id_\mathrm{c} <' id_\mathrm{e}$ *of* $<$*. However, this is hard to see, because each of fr, it, 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, such as* (it, fr, fr, it, jp, jp)*.*

In the following sections we study the computational complexity of the `POSS` and `CERT` problems, for multiple fragments of our language.

4. General Complexity Results

We have defined the PosRA and PosRA$^{\mathrm{acc}}$ 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$^{\mathrm{acc}}$, 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$^{\mathrm{acc}}$). We summarize the complexity results of Sections 4–6 in Table 1.

In this section, we state our main complexity results and prove the corresponding upper bounds. Lower bounds will be implied by more precise results

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

Table 1: Summary of complexity results for possibility and certainty

	Query	Restr. on accum.	Input po-relations	Complexity	
POSS	PosRA/PosRA$^{\text{acc}}$	—	arbitrary	NP-c.	(Thm. 18)
CERT	PosRA$^{\text{acc}}$	—	arbitrary	coNP-c.	(Thm. 19)
CERT	PosRA	—	arbitrary	PTIME	(Thm. 32)
POSS	PosRA$_{\text{LEX}}$	—	width $\leqslant k$	PTIME	(Thm. 20)
POSS	PosRA$_{\text{DIR}}$	—	totally ordered	NP-c.	(Thm. 22)
POSS	PosRA$_{\text{no}\times}$	—	ia-width or width $\leqslant k$	PTIME	(Thm. 24)
POSS	PosRA$_{\text{LEX}}$/PosRA$_{\text{DIR}}$	—	1 total. ord., 1 unord.	NP-c.	(Thm. 31)
CERT	PosRA$^{\text{acc}}$	cancellative	arbitrary	PTIME	(Thm. 32)
POSS	PosRA$^{\text{acc}}$	finite and pos.-invar.	totally ordered	NP-c.	(Thm. 37)
CERT	PosRA$^{\text{acc}}$	finite and pos.-invar.	totally ordered	coNP-c.	(Thm. 42)
both	PosRA$^{\text{acc}}_{\text{LEX}}$	finite	width $\leqslant k$	PTIME	(Thm. 43)
both	PosRA$^{\text{acc}}_{\text{no}\times}$	finite and pos.-invar.	ia-width or width $\leqslant k$	PTIME	(Thm. 45)
POSS	PosRA$^{\text{acc}}_{\text{no}\times}$	pos.-invar.	unordered	NP-c.	(Thm. 47)

that will be established in Sections 5 and 6.

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

Theorem 18. *The POSS problem is in NP for any fixed 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. To show that POSS is in NP, evaluate the query without accumulation in PTIME using Proposition 2, yielding a po-relation Γ. Now, guess a total order of Γ, checking in PTIME that it is compatible with the comparability relations of Γ. If there is no accumulation function, then check that it achieves the candidate result. Otherwise, evaluate the accumulation (in PTIME as the accumulation operator is PTIME-evaluable), and check that the correct result is obtained. This shows that POSS is in NP for PosRA and PosRA$^{\text{acc}}$ queries. The NP-hardness will follow from stronger results that will be shown later: Theorem 22 for PosRA and Theorem 37 for PosRA$^{\text{acc}}$. □

A different route to prove the NP-hardness of POSS is to use existing work [14] about the complexity of the so-called *shuffle problem*: given a string w and a tuple of strings $s_1, \ldots, s_n$ on the fixed alphabet $A = \{a, b\}$, decide whether there is an interleaving of $s_1, \ldots, s_n$ which is equal to w. It is easy to see that there is a reduction from the shuffle problem to the POSS problem, by representing each string s_i as a totally ordered relation L_i of tuples labeled a and b that code the string, letting Γ be the po-relation defined as the union of the L_i, and checking if the totally ordered relation that codes w is a possible world of the identity PosRA query on the po-relation Γ. Hence, as the shuffle problem is NP-hard [14], we deduce that POSS is NP-hard. However, this approach will not suffice to derive the stronger NP-hardness results which we prove in the sequel.

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

Theorem 19. *The CERT problem is in coNP for any fixed PosRA$^{\text{acc}}$ query, and there is a PosRA$^{\text{acc}}$ query for which it is coNP-complete.*

Proof. The co-NP upper bound is proved using precisely the same reasoning applied to the NP upper bound for POSS, except that we now guess an order that achieves a result *different* from the candidate result. The hardness result for CERT and $\text{PosRA}^{\text{acc}}$ is presented (in a slightly stronger form) as Theorem 42 in the sequel. □

For PosRA queries, we will show that CERT is in PTIME. This will follow from a stronger result that we will prove in the sequel (Theorem 32): CERT is in PTIME for $\text{PosRA}^{\text{acc}}$ queries that perform accumulation in a cancellative monoid.

Practical implications. We now discuss some implications of the results highlighted in Table 1 on the implementation of the algebra on top of, say, a SQL database engine. First, recall Proposition 2: computing the result of a query, as a po-relation, is in PTIME in the size of the input database, and can thus reasonably be implemented. Second, thanks to Theorem 32, since CERT is in PTIME for PosRA, it should also be possible to implement certainty tests efficiently. However, Theorem 18 shows that possibility tests are prohibitive to implement in all generality.

However, in a practical context, input relations are usually not arbitrary po-relations: it makes sense to assume in many scenarios that input relations are either totally ordered (say, because they are ordered by their primary key, or by an explicit ORDER BY construct) or unordered (because no specific ordering has been chosen). In this case, we have two ways to ensure that the possibility problem is tractable: either we only allow totally ordered po-relations as input and then Theorem 20 (in Section 5) shows that possibility is tractable if the only product operator allowed is $\times_{\text{LEX}}$; or we allow both totally ordered and unordered po-relations as input, but then only queries with no product are tractable for possibility tests (which, arguably, considerably limits the expressive power).

When moving to $\text{PosRA}^{\text{acc}}$, the picture is similar, but we need additional properties of the accumulation function to ensure that the possibility and certainty problems are tractable (depending on the cases, it should be cancellative, finite, or position-invariant).

We next identify further tractable cases. In the following section, we study PosRA queries: we focus on POSS, as we know that CERT is always in PTIME for such queries. In Section 6, we turn to $\text{PosRA}^{\text{acc}}$.

5. Tractable Cases for POSS on PosRA Queries

We have stated a general NP-hardness result for POSS with PosRA queries. We next show that tractability may be achieved if we both restrict the allowed operators and bound some order-theoretic parameters of the input po-database, e.g., its width. Recall that $\text{PosRA}_{\text{LEX}}$ (respectively, $\text{PosRA}_{\text{DIR}}$) denotes the fragment of PosRA that disallows $\times_{\text{DIR}}$ (respectively, $\times_{\text{LEX}}$).

5.1. (Almost) Totally Ordered Inputs

We start by the natural case where we assume that the width of all input po-relations is bounded by a constant. This assumption is a common practical case: it covers the case where all input 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 arises because of the integration query. The assumption also covers the case of po-relations that are totally ordered except for a few "tied" data items at each level. Recall that the query result can still have exponentially many possible worlds under this assumption, e.g., when taking the union of two totally ordered relations. In a sense, the $\times_{\texttt{DIR}}$ operator is the one introducing the most uncertainty and "unorderedness" in the result, so we consider the fragment $\text{PosRA}_{\texttt{LEX}}$ of PosRA queries without $\times_{\texttt{DIR}}$, and show the following result.

Theorem 20. *For any fixed $k \in \mathbb{N}$ and fixed $\text{PosRA}_{\texttt{LEX}}$ query Q, the* POSS *problem for Q is in PTIME when all po-relations of the input po-database have width $\leqslant k$.*

To show this result, letting D be the input po-database, we can use Proposition 2 to evaluate $\Gamma := Q(D)$ in PTIME. Recall that we have previously shown Lemma 4 on $\text{PosRA}_{\texttt{LEX}}$, so we know that the width of the po-relation Γ is constant: it only depends on k and Q, but not on D. Hence, to show Theorem 20, it suffices to show the following.

Lemma 21. *For any constant $k \in \mathbb{N}$, we can determine in PTIME, for any po-relation Γ with width $\leqslant k$ and list relation L, whether $L \in pw(\Gamma)$.*

Let us prove this lemma and conclude the proof of Theorem 20.

Proof. Let $\Gamma = (ID, T, <)$ be the po-relation of width $k' \leqslant k$, and let $P = (ID, <)$ be its underlying poset. We use Dilworth's theorem [7, 6] to compute in PTIME a chain partition $ID = \Lambda_1 \sqcup \cdots \sqcup \Lambda_{k'}$ of P. For $1 \leqslant i \leqslant k'$, we write $n_i := |\Lambda_i|$, we write $\Lambda_i[j]$ for $1 \leqslant j \leqslant n_i$ to denote the j-th element of Λ_i, and for $0 \leqslant j \leqslant n_i$, we write $\Lambda_i^{\leqslant j}$ to denote the first j elements of the chain Λ_i, formally, $\Lambda_i^{\leqslant j} := \{\Lambda_i[j'] \mid 1 \leqslant j' \leqslant j\}$. In particular, $\Lambda_i^{\leqslant 0} = \emptyset$ and $\Lambda_i^{n_i} = \Lambda_i$.

We now consider all vectors $\mathbf{m}$ of the form $(m_1, \ldots, m_{k'})$, with $0 \leqslant m_i \leqslant n_i$ for each $1 \leqslant i \leqslant k'$. There are polynomially many such vectors, more specifically at most $|\Gamma|^k$ of them (recall that k is a constant). To each such vector $\mathbf{m}$ we associate the subset $s(\mathbf{m})$ of P consisting of $\bigsqcup_{i=1}^{k'} \Lambda_i^{\leqslant m_i}$.

We call such a vector $\mathbf{m}$ *sane* if $s(\mathbf{m})$ is an order ideal. Note that this is not always the case: while $s(\mathbf{m})$ is always an order ideal of the subposet of the comparability relations within the chains, it may not be an order ideal of P overall because of the additional comparability relations across the chains. For each vector $\mathbf{m}$, we can check in PTIME whether it is sane: simply materialize $s(\mathbf{m})$ and check that it is an ideal by considering each of the $\leqslant |P|^2$ comparability relations.

By definition, for each sane vector $\mathbf{m}$, we know that $s(\mathbf{m})$ is an ideal. We now observe that the converse is also true: for every ideal S of P, there is a sane vector $\mathbf{m}$ such that $s(\mathbf{m}) = S$. To see why, consider any ideal S, and determine for each $1 \leqslant i \leqslant k'$ the last element of the chain Λ_i which is in S: let $m_i := 1 \leqslant i \leqslant n_i$ be the position of this element in Λ_i, where we set $m_i := 0$ if S contains no element of Λ_i. We know that S does not include any element of Λ_i at a position later than m_i, and because Λ_i is a chain it must include all elements before m_i; in other words, we have $S \cap \Lambda_i = \Lambda_i^{\leqslant m_i}$. As $(\Lambda_i)_{1 \leqslant i \leqslant k'}$ is a chain partition of P, this uniquely determines S. Thus we have indeed $S = s(\mathbf{m})$, and the fact that $s(\mathbf{m})$ is sane is witnessed by S.

We now use a dynamic algorithm to compute, for each sane vector $\mathbf{m}$, a Boolean denoted $t(\mathbf{m})$ which is true iff there is a topological sort of $s(\mathbf{m})$ whose label is the prefix of the candidate possible world L having length $|s(\mathbf{m})| = \sum_{i=1}^{k'} m_i$. We extend the function t to arbitrary vectors by setting $t(\mathbf{m}) := 0$ whenever $\mathbf{m}$ is not sane. Specifically, the base case is that $t(0, \ldots, 0) := \text{true}$, because the empty ideal trivially achieves the empty prefix. To define the induction case, let us denote by e_i for $1 \leqslant i \leqslant k'$ the vector consisting of $n-1$ zeros and a 1 at position i. Now, for each sane vector $\mathbf{m}$, we have:

$$t(\mathbf{m}) := \bigvee_{\substack{1 \leqslant i \leqslant k' \\ m_i > 0}} \left(\left(T(\Lambda_i[m_i]) = L\left[\sum_{i'=1}^{k'} m_{i'} \right] \right) \wedge t(\mathbf{m} - e_i) \right)$$

where L is the candidate possible world and where "$-$" denotes the componentwise difference on vectors. It is clear that $t(\mathbf{m})$ is correct by induction: the key argument is that, for any sane vector $\mathbf{m}$, any linear extension of $s(\mathbf{m})$ must finish by enumerating one of the maximal elements of $s(\mathbf{m})$, that is, $\Lambda_i[m_i]$ for some $1 \leqslant i \leqslant k'$ such that $m_i > 0$: and then the linear extension achieves the prefix of L of length $|s(\mathbf{m})|$ iff the following two conditions are true: (i.) the label by T of the last element in the linear extension must be the label of element of L at position $|s(\mathbf{m})|$; and (ii.) $\mathbf{m} - e_i$ must be a sane vector such that the restriction of the linear extension to $s(\mathbf{m} - e_i)$ achieves the prefix of L of length $|s(\mathbf{m} - e_i)|$ which by induction was computed as $t(\mathbf{m} - e_i)$.

It is now clear that we can compute all $t(\mathbf{m})$ in PTIME by a dynamic algorithm: we enumerate the vectors (of which there are polynomially many) in lexicographical order, and computing their image by t in PTIME according to the equation above, from the base case $t(0, \ldots, 0) = \varepsilon$ and from the previously computed values of t, recalling that $t(\mathbf{m}') := 0$ whenever $\mathbf{m}'$ is not sane. Now, $t(n_1, \ldots, n_{k'})$ is true iff Γ has a linear extension achieving L, so we have indeed solved the POSS problem for Γ and L in PTIME, concluding the proof. □

We have now shown Theorem 20 and established tractability for POSS with $\text{PosRA}_{\text{LEX}}$ queries on po-databases of bounded width. We will show in Theorem 43 that this proof technique further extends to queries with accumulation, under some assumptions over the accumulation function.

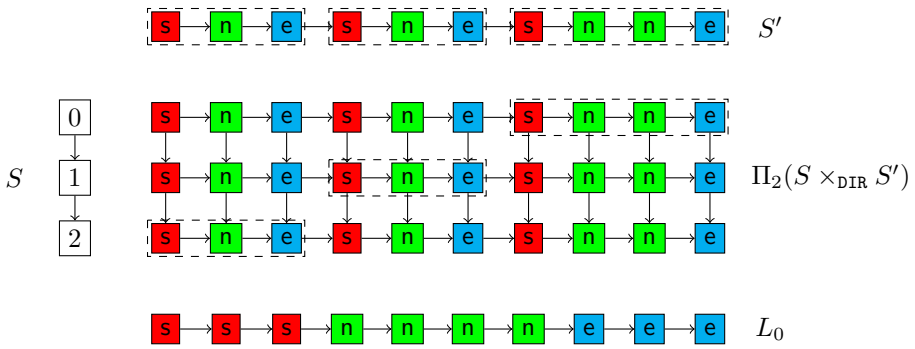


Figure 4: Example for the proof of Theorem 22.

We next show that our tractability result only holds for $\text{PosRA}_{\texttt{LEX}}$. Indeed, if we allow $\times_{\texttt{DIR}}$, then POSS is hard on totally ordered po-relations, even if we disallow $\times_{\texttt{LEX}}$. This result implies the general NP-hardness result on POSS that we stated earlier (Theorem 18) for queries without accumulation.

Theorem 22. *There is a $\text{PosRA}_{\texttt{DIR}}$ query for which the POSS problem is NP-complete even when input po-databases consist only of totally ordered po-relations.*

Proof. We reduce from the NP-hard UNARY-3-PARTITION problem [15]: given $3m$ integers $E = (n_1, \ldots, n_{3m})$ written in unary (not necessarily distinct) and a number B, decide if the integers can be partitioned in triples such that the sum of each triple is B. We reduce an instance $\mathcal{I} = (E, B)$ of UNARY-3-PARTITION to a POSS instance in PTIME. We fix $\mathcal{D} := \mathbb{N} \sqcup \{\mathsf{s}, \mathsf{n}, \mathsf{e}\}$, with s, n and e standing for *start*, *inner*, and *end*.

Let D be the po-database which interprets the relation name S by the totally ordered po-relation $[\leqslant 3m-1]$, and the relation name S' by the totally ordered po-relation constructed from the instance $\mathcal{I}$ as follows: for $1 \leqslant i \leqslant 3m$, consider the concatenation of one tuple id^i_1 with value s, n_i tuples id^i_j (with $2 \leqslant j \leqslant n_i+1$) with value n, and one tuple $id^i_{n_i+2}$ with value e, and define the interpretation of S' by concatenating the $3m$ sequences of length $n_i + 2$. Consider the query $Q := \Pi_2(S \times_{\texttt{DIR}} S')$, where Π_2 projects to the attribute of the relation S'. See Figure 4 for an illustration with $E = (1, 1, 2)$ and $B = 4$.

We define the candidate possible world L as the list relation $L := L_1 L' L_2$, with L_1, L', and L_2 defined as follows.

- L_1 is a list relation defined as the concatenation, for $1 \leqslant i \leqslant 3m$, of $3m - i$ copies of the following sublist: one tuple with value s, n_i tuples with value n, and one tuple with value e.
- L_2 is a list relation defined like L_1, except that $3m - i$ is replaced by $i - 1$.
- L_0 is the list relation consisting of three tuples with value s, B tuples with value n, three tuples with value e. See Figure 4 for an illustration of L_0.

- L' is the list relation defined as the concatenation of m copies of L_0.

We now consider the POSS instance that asks whether L is a possible world of the query Q on the po-database D. We claim that this POSS instance is positive iff the original UNARY-3-PARTITION instance $\mathcal{I}$ is positive. As the reduction process described above is clearly PTIME, the only thing left to prove Theorem 22 is to show this claim, which we now do.

Denote by Γ' the po-relation obtained by evaluating $Q(D)$, and note that all tuples of Γ' have value in $\{\mathsf{s}, \mathsf{n}, \mathsf{e}\}$. For $0 \leqslant k \leqslant |L_1|$, we write $L_1^{\leqslant k}$ for the prefix of L_1 of length k. We say that $L_1^{\leqslant k}$ is a *whole prefix* if either $k = 0$ (that is, the empty prefix) or the k-th symbol of L_1 has value e. We say that a linear extension L'' of Γ' *realizes* $L_1^{\leqslant k}$ if the sequence of its k-th first values is $L_1^{\leqslant k}$, and that it realizes L_1 if it realizes $L_1^{\leqslant |L_1|}$. When L'' realizes $L_1^{\leqslant k}$, we call the *matched* elements the elements of Γ' that occur in the first k positions of L'', and say that the other elements are *unmatched*. For $1 \leqslant i \leqslant 3m$, we call the *$i$-th row* of Γ' the elements whose first component before projection was $i - 1$: note that, for each i, the po-relation Γ' imposes a total order on the i-th row. We define the *row-i matched elements* to refer to the elements on row-i that are matched, and define analogously the *row-i unmatched elements*.

We first observe that for any linear extension L'' realizing $L_1^{\leqslant k}$, for all i, writing the i-th row as $id'_1 < \ldots < id'_{|S'|}$, the unmatched elements must be all of the form id'_j for $k_i < j \leqslant |S'|$ for some $0 \leqslant k_i \leqslant |S'|$, i.e., they must be a prefix of the total order of the i-th row. Indeed, if they did not form a prefix, then some order constraint of Γ' would have been violated when enumerating L''. Further, by cardinality we clearly have $\sum_{i=1}^{3m} k_i = k$.

Second, when a linear extension L'' of Γ' realizes $L_1^{\leqslant k}$, we say that we are in a *whole situation* for k if for all i, either the first row-i unmatched element id'_{k_i+1} has value s or there are no row-i unmatched elements (and we write $k_i := |S'|$). When we are in a whole situation for k, the condition on k_i means by definition that we must have $k_i = \sum_{j=1}^{l_i}(n_j + 2)$ for some $1 \leqslant l_i \leqslant 3m$; in this case, letting S_i be the multiset of the n_j for $1 \leqslant j \leqslant l_i$, we call S_i the bag of *row-i consumed integers* at k. The *row-i remaining integers* at k are $E \setminus S_i$, where we see E as a multiset and define the difference operator on multisets by subtracting the multiplicities in S_i to the multiplicities in E.

We now prove the following claim: for any linear extension of Γ' realizing L_1, we are in a whole situation for $|L_1|$, and the multiset union $\biguplus_{1 \leqslant i \leqslant 3m} S_i$ of the row-i consumed integers at k is equal to the multiset obtained by repeating $3m - i$ times the integer n_i of E for all $1 \leqslant i \leqslant 3m$.

We prove the first part of the claim by showing it for all whole prefixes $L_1^{\leqslant k}$, by induction on k. It is certainly the case for $L_1^{\leqslant 0}$ (the empty prefix). Now, assuming that it holds for prefixes of length up to l, to realize a whole prefix $L^{\leqslant l'}$ with $l' > l$, we must first realize a strictly shorter whole prefix $L^{\leqslant l''}$ with $l'' \leqslant l$ (take it to be of maximal length), so by induction hypothesis we are in a whole situation for l'' when realizing $L^{\leqslant l''}$. Now to realize the whole prefix $L^{\leqslant l'}$ having realized the whole prefix $L^{\leqslant l''}$, by construction of L_1, the sequence L'' of

additional values to realize is s, a certain number of n's, and e. It is now clear that this must bring us from a whole situation to a whole situation: since there is only one s in L'', there is only one row such that an s value becomes matched; now, to match the additional n's and e, only the elements of this particular row can be used, as any first unmatched element (if any) of all other rows is s, and we must use the sequence of n-labeled elements followed by the e-labeled element of the row. Hence the first part of the claim is proved.

To prove the second part of the claim, observe that whenever we go from a whole prefix to a whole prefix by additionally matching s, n_j times n, and e, then we add to S_i the integer n_j. So the claim holds by construction of L_1.

A similar argument shows that for any linear extension L'' of Γ' whose first $|L_1|$ tuples achieve L_1 and whose last $|L_2|$ tuples achieve L_2, for each $1 \leqslant i \leqslant 3m$, extending the definition of the row-i unmatched elements to refer to the elements that are matched neither to L_1 nor to L_2, these elements must form a contiguous sequence id'_j with $k_i < j < m_i$ for some $0 \leqslant k_i < m_i \leqslant |S'| + 1$: here k_i refers to the last element of row i matched to L_1 (or 0 if none are), and m_i to the first element of row i matched to L_2 (or $|S'| + 1$ if none are). In addition, if we have $k_i < m_i - 1$, then id'_{k_i} has value e and id'_{m_i} has value s, and the unmatched values (whose definition is extended in an analogous fashion) are a multiset corresponding exactly to the elements $n_1, \ldots, n_{3m}$: indeed, each integer n_i of E is matched $3m-i$ times within L_1 and $i-1$ times in L_2, so $3m-i+i-1 = 3m-1$ times overall, whereas it occurs $3m$ times in the grid. So the unmatched elements when having read L_1 (at the beginning) and L_2 (at the end) are formed of $3m$ sequences, of length $n_i + 2$ for $1 \leqslant i \leqslant 3m$, of the form s, n_i times n, and e: each of the $3m$ sequences is totally ordered (as it occurs as consecutive elements in some row), and there is a certain order relation across the sequences depending on the rows where they are: the comparability relations exist across sequences that are on the same row, or that are in different rows but where comparability holds by definition of $\times_{\texttt{DIR}}$.

Observe now that there is a way to achieve L_1 and L_2 while ensuring that there are no order constraints across the sequences of unmatched elements, i.e., the only order constraints within the unmatched elements are those given by the total order on each sequence. To do so, we achieve L_1 by picking the following, in that order: for $1 \leqslant j \leqslant 3m$, for $1 \leqslant i \leqslant 3m - j$, pick the first $n_j + 2$ unmatched tuples of row i. Similarly, to achieve L_2 at the end, we can pick the following, in *reverse* order: for $3m \geqslant j \geqslant 1$, for $3m \geqslant i \geqslant 3m-j+1$, the last n_j+2 unmatched tuples of row i. When we pick elements this way, the unmatched elements are $3m$ lists (one for each row, with that of row i being s, n_i times n and e, for all i) and there are no order relations across sequences. We let Γ be the sub-po-relation of Γ' that consists of exactly these unmatched elements: it is illustrated in Figure 4 as the elements of the grid that are in the dashed rectangles. Formally, Γ is the parallel composition of $3m$ totally ordered po-relations which we will call Γ_i for $1 \leqslant i \leqslant 3m$: the elements of Γ_i consist of an element labeled s followed by n_i elements labeled n and one element labeled e.

We now claim that for any list relation L'', the concatenation $L_1 L'' L_2$ is a possible world of Γ' if and only if L'' is a possible world of Γ. The "if" direction

was proved with the construction above, and the "only if" holds because Γ is the *least constrained* possible po-relation for the unmatched sequences: recall that the only comparability relations that it contains are those on the sequences of unmatched elements, which are known to be total orders. Hence, to prove our original claim, it only remains to show that the UNARY-3-PARTITION instance $\mathcal{I}$ is positive iff L' is a possible world of Γ, which we now do.

For the forward direction, we show that, if $\mathcal{I}$ is a positive instance of UNARY-3-PARTITION, then there is a linear extension $<'$ of $<$ which witnesses that $L' \in pw(\Gamma)$. Indeed, consider a 3-partition $\mathbf{p} = (p_1^i, p_2^i, p_3^i)$ for $1 \leqslant i \leqslant m$, with $n_{p_1^i} + n_{p_2^i} + n_{p_3^i} = B$ for all $1 \leqslant i \leqslant m$, and each integer of $\{1, \ldots, 3m\}$ occurring exactly once in $\mathbf{p}$. We can realize L' from $\mathbf{p}$ by picking successively the following for $1 \leqslant i \leqslant m$ to realize L_0: the three s-labeled elements of the po-relations $\Gamma_{p_q^i}$ for $1 \leqslant q \leqslant 3$, then the n-labeled elements of these same po-relations (this is B tuples in total, because $\mathbf{p}$ is a solution to $\mathcal{I}$), and last the three e-labeled elements of these po-relations.

For the backward direction, we show that, if there is a linear extension $<'$ of $<$ which witnesses that $L' \in pw(\Gamma)$, then we can build a 3-partition $\mathbf{p} = (p_1^i, p_2^i, p_3^i)$ for $1 \leqslant i \leqslant m$ which satisfies the conditions above. To see why, we first observe that, for each $1 \leqslant i \leqslant m$, considering the i-th occurrence of the sublist L_0 in L', there must be three distinct values p_1^i, p_2^i, p_3^i, such that the elements which occur in $<'$ at the positions of the value n in this occurrence of L_0 are precisely the n-labeled elements of the po-relations $\Gamma_{p_1^i}$, $\Gamma_{p_2^i}$, and $\Gamma_{p_3^i}$. Indeed, we show this claim for increasing values of i, from $i = 1$ to $i = m$. Just before we consider each occurrence of L_0, and just after we have considered it, we will ensure the invariant that, for all $1 \leqslant i \leqslant 3m$, either all elements of Γ_i have been enumerated or none have: this invariant is clearly true initially because nothing is enumerated yet. Now, considering the i-th occurrence of L_0 for some $1 \leqslant i \leqslant m$, we define p_1^i, p_2^i, p_3^i, such that the elements s^3 in this occurrence of L_0 are mapped to the s-labeled elements of $\Gamma_{p_1^i}$, $\Gamma_{p_2^i}$, and $\Gamma_{p_3^i}$: they must indeed be mapped to such elements because they are the only ones with value s. Now, the n-labeled elements of these three po-relations can all be enumerated (indeed, we have just enumerated the s-labeled elements that precede them), and they are the only elements with value n that can be enumerated, thanks to the invariant: the others either have already been enumerated or have a predecessor with value s that has not been enumerated yet. Further, all elements of this form must be enumerated, because this is the only possible way for us to finish matching L_0 and enumerate three elements with value e, namely, those of the three po-relations $\Gamma_{p_1^i}$, $\Gamma_{p_2^i}$, and $\Gamma_{p_3^i}$: this uses the invariant again to justify that they are the only elements with value e that can be enumerated at this stage. We are now done with the i-th occurrence of L_0, and clearly the invariant is satisfied on the result, because the elements that we have enumerated while matching this occurrence of L_0 are all the elements of $\Gamma_{p_1^i}$, $\Gamma_{p_2^i}$, and $\Gamma_{p_3^i}$.

Now that we have defined the 3-partition $\mathbf{p}$, it is clear by definition of a linear extension that each position $1 \leqslant i \leqslant 3m$, i.e., each number occurrence in E, must occur exactly once in $\mathbf{p}$. Further, as $<'$ achieves L_0, by considering

each occurrence of L_0, we know that, for $1 \leqslant i \leqslant m$, we have $p_1^i + p_2^i + p_3^i = B$. Hence, $\mathbf{p}$ witnesses that $\mathcal{I}$ is a positive instance to the UNARY-3-PARTITION problem.

Hence, it is indeed the case that $\mathcal{I}$ is a positive UNARY-3-PARTITION instance iff $L' \in pw(\Gamma)$, which is the case iff $L_1 L' L_2$ is a possible world of Γ', i.e., iff L is a possible world of $Q(D)$. This establishes the correctness of the reduction for PosRA, showing that the POSS problem for PosRA queries is NP-hard. □

5.2. Disallowing Both Products

We have shown the tractability of POSS without the $\times_{\text{DIR}}$ operator, when the input po-relations are assumed to have bounded width. We now study the fragment $\text{PosRA}_{\text{no}\times}$ without both kinds of product, and show that this POSS is tractable for this fragment even for more general input po-relations. Specifically, we will allow input po-relations that are almost totally ordered, i.e., have bounded *width*; and we will also allow input po-relations that are almost unordered, which we measure using a new order-theoretic notion of *ia-width*. The idea of ia-width is to decompose the relation in classes of indistinguishable sets of incomparable elements.

Definition 23. *Given a poset $P = (ID, <)$, a subset $A \subseteq ID$ is an* antichain *if there are no $x, y \in A$ such that $x < y$. It is an* indistinguishable set *(or an* interval *[16]) if, for all $x, y \in A$ and $z \in ID \backslash A$, we have $x < z$ iff $y < z$, and $z < x$ iff $z < y$. It is an* indistinguishable antichain *if it is both an antichain and an indistinguishable set.*

An indistinguishable antichain partition *(ia-partition) of P is a partition of ID into indistinguishable antichains. The* cardinality *of the partition is the number of antichains. The* ia-width *of P is the cardinality of its smallest ia-partition. The* ia-width *of a po-relation is that of its underlying poset, and the* ia-width *of a po-database is the maximal ia-width of its po-relations.*

Hence, any po-relation Γ has ia-width at most $|\Gamma|$, with the trivial ia-partition consisting of singleton indistinguishable antichains, and unordered po-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 they consist of objects from a constant number of types that are ordered based only on some order on the types.

We can now state our tractability result when disallowing both kinds of products, and allowing both bounded-width and bounded-ia-width relations. For instance, this result allows us to combine sources whose order is fully unknown or irrelevant, with sources that are completely ordered (or almost totally ordered).

Theorem 24. *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 each po-relation of the input po-database has either ia-width $\leqslant k$ or width $\leqslant k$.*

To prove this result, we start by making a simple observation.

Lemma 25. *Any $PosRA_{\text{no}\times}$ query Q can be equivalently rewritten as a union of projections of selections of a constant number of input relations and constant relations.*

Proof. For the semantics that we have defined for operators, it it easy to show that selection commutes with union, selection commutes with projection, and projection commutes with union. Hence, we can perform the desired rewriting. □

We can thus rewrite the input query using this lemma. The idea is that we will evaluate the query in PTIME using Proposition 2, argue that the width bounds are preserved using Lemma 4, and compute a chain partition of the relations using Dilworth's theorem. Let us first show an analogue of Lemma 4 for the new notion of ia-width.

Lemma 26. *Let $k \geqslant 2$ and Q be a $PosRA_{\text{no}\times}$ query. For any po-database D of ia-width $\leqslant k$, the po-relation $Q(D)$ has ia-width $\leqslant \max(k, q) \times |Q|$, where q denotes the largest value such that $[\leqslant q]$ appears in Q.*

Proof. We first show by induction on Q that the ia-width of the query output can be bounded by a function of k. We show the base cases.

- The input relations have ia-width at most k.
- The constant relations have ia-width $\leqslant q$ with the trivial ia-partition consisting of singleton classes.

We then show the induction step.

- Projection clearly does not change ia-width.
- Selection may only decrease the ia-width. Indeed, consider an ia-partition of the input po-relation, apply the selection to each class, and remove the classes that became empty. The number of classes has not increased, and it is clear that the result is still an ia-partition of the output po-relation.
- The union of two relations with ia-width k_1 and k_2 has ia-width at most $k_1 + k_2$. Indeed, we can obtain an ia-partition for the union as the union of ia-partitions for the input relations.

Second, we see that the bound $\max(k, q) \times |Q|$ on the ia-width of $Q(D)$ is clearly correct, because the base cases have ia-width $\leqslant \max(k, q)$ and the worst operators are unions, which amount to summing the ia-width bounds on all inputs, of which there are $\leqslant |Q|$. So we have shown the desired bound. □

We next show that, like chain partitions for bounded-width po-relations, we can efficiently compute an ia-partition for a bounded-ia-width po-relation.

Proposition 27. *The ia-width of any poset and a corresponding ia-partition can be computed in PTIME.*

To show this result, we need two preliminary observations about indistinguishable antichains.

Lemma 28. *For any poset $(ID, <)$ and indistinguishable antichain A, any $A' \subseteq A$ is an indistinguishable antichain.*

Proof. Clearly A' is an antichain because A is. We show that it is an indistinguishable set. Let $x, y \in A'$ and $z \in ID \backslash A'$, and show that $x < z$ implies $y < z$ (the other three implications are symmetric). If $z \in ID \backslash A$, then we conclude because A is an indistinguishable set. If $z \in A \backslash A'$, then we conclude because, as A is an antichain, z is incomparable both to x and to y. □

Lemma 29. *For any poset $(ID, <)$ and indistinguishable antichains $A_1, A_2 \subseteq ID$ such that $A_1 \cap A_2 \neq \emptyset$, the union $A_1 \cup A_2$ is an indistinguishable antichain.*

Proof. We first show that $A_1 \cup A_2$ is an indistinguishable set. Let $x, y \in A_1 \cup A_2$ and $z \in ID \backslash (A_1 \cup A_2)$, assume that $x < z$ and show that $y < z$ (again the other three implications are symmetric). As A_1 and A_2 are indistinguishable sets, this is immediate unless $x \in A_1 \backslash A_2$ and $y \in A_2 \backslash A_1$, or vice-versa. We assume the first case as the second one is symmetric. Consider $w \in A_1 \cap A_2$. As $x < z$, we know that $w < z$ because A_1 is an indistinguishable set, so that $y < z$ because A_2 is an indistinguishable set, which proves the desired implication.

Second, we show that $A_1 \cup A_2$ is an antichain. Proceed by contradiction, and let $x, y \in A_1 \cup A_2$ such that $x < y$. As A_1 and A_2 are antichains, we must have $x \in A_1 \backslash A_2$ and $y \in A_2 \backslash A_1$, or vice-versa. Assume the first case, the second case is symmetric. As A_1 is an indistinguishable set, letting $w \in A_1 \cap A_2$, as $x < y$ and $x \in A_1$, we have $w < y$. But $w \in A_2$ and $y \in A_2$, which is impossible because A_2 is an antichain. We have reached a contradiction, so we cannot have $x < y$. Hence, $A_1 \cup A_2$ is an antichain, which concludes the proof. □

We can now show Proposition 27.

Proof. Start with the trivial partition in singletons (which is an ia-partition), and for every pair of items, see if their current classes can be merged (i.e., merge them, check in PTIME if it is an antichain, and if it is an indistinguishable set, and undo the merge if it is not). Repeat the process while it is possible to merge classes (i.e., at most linearly many times). This greedy process concludes in PTIME and yields an ia-partition $\mathbf{A}$. Let n be its cardinality.

Now assume that there is an ia-partition $\mathbf{A}'$ of cardinality $m < n$. There has to be a class A' of $\mathbf{A}'$ which intersects two different classes $A_1 \neq A_2$ of the greedy ia-partition $\mathbf{A}$, otherwise $\mathbf{A}'$ would be a refinement of $\mathbf{A}$ so we would have $m \geqslant n$. Now, by Lemma 29, $A \cup A_1$ and $A \cup A_2$, and hence $A \cup A_1 \cup A_2$, are indistinguishable antichains. By Lemma 28, this implies that $A_1 \cup A_2$ is an indistinguishable antichain. Now, when constructing the greedy ia-partition $\mathbf{A}$, the algorithm has considered one element of A_1 and one element of A_2, attempted to merge the classes A_1 and A_2, and, since it has not merged them in $\mathbf{A}$, the union $A_1 \cup A_2$ cannot be an indistinguishable antichain. We have reached a contradiction, so we cannot have $m < n$, which concludes the proof. □

We have shown the preservation of ia-width bounds through selection, projection, and union (Lemma 26), and shown how to compute an ia-partition in PTIME (Proposition 27). Let us now return to the proof of Theorem 24. We use Lemma 25 to rewrite the query to a union of projection of selections. We evaluate the selections and projections in PTIME by Proposition 2. As union is clearly associative and commutative, we evaluate the union of relations of width $\leqslant k$, yielding Γ, and the union of those of ia-width $\leqslant k$, yielding Γ'. The first result Γ has bounded width thanks to Lemma 4, and we can compute a chain partition of it in PTIME using Dilworth's theorem. The second result has bounded ia-width thanks to Lemma 26, and we can compute an ia-partition of it in PTIME using Proposition 27. Hence, to show Theorem 24, it suffices to show the following strengthening of Lemma 21.

Lemma 30. *For any constant $k \in \mathbb{N}$, we can determine in PTIME, for any input po-relation Γ with width $\leqslant k$, input po-relation Γ' with ia-width $\leqslant k$, and list relation L, whether $L \in pw(\Gamma \cup \Gamma')$.*

Proof. We first show the result when assuming that Γ is empty, and will later return to the general case. Let $\mathbf{A} = (A_1, \ldots, A_k)$ be an ia-partition of width k of $\Gamma' = (ID, T, <)$, which can be computed in PTIME by Proposition 27. We assume that the length of the candidate possible world L is $|ID|$, as we can trivially reject otherwise.

For any linear extension $<'$ of Γ', we define the *finishing order* of $<'$ as the permutation π of $\{1, \ldots, k\}$ obtained by considering, for each class A_i of $\mathbf{A}$, the largest position $1 \leqslant n_i \leqslant |ID|$ in $<'$ to which an element of A_i is mapped, and sorting the class indexes in ascending order according to this largest position. We say we can realize L with finishing order π if there is a linear extension of Γ' that realizes L and whose finishing order is π. Hence, it suffices to check, for every possible permutation π of $\{1, \ldots, k\}$, whether L can be realized from Γ' with finishing order π: this does not make the complexity worse because the number of finishing orders depends only on k and not on Γ', so it is constant. (Note that the order relations across classes may imply that some finishing orders are impossible to realize altogether.)

We now claim that to determine whether L can be realized with finishing order π, the following greedy algorithm works. Read L linearly. At any point, maintain the set of elements of Γ' that have already been enumerated (distinguish the *used* and *unused* elements; initially all elements are unused), and distinguish the classes of $\mathbf{A}$ in three kinds: the *exhausted classes*, where all elements are used; the *open classes*, the ones where some elements are unused and all ancestor elements outside of the class are used; and the *blocked classes*, where some ancestor element outside of the class is not used. Initially, the open classes are those which are roots in the poset obtained from the underlying poset of Γ' by taking the quotient by the equivalence relation induced by $\mathbf{A}$; and the other classes are blocked.

When reading a value t from L, consider all open classes. If none of these classes have an unused element with value t, reject, i.e., conclude that we cannot realize L as a possible world of Γ' with finishing order π. Otherwise, take the

open class that comes first in the finishing order, and use an arbitrary suitable element from it. Update the class to be *exhausted* if it is: in this case, check that the class was the next one in the finishing order π (and reject otherwise), and update from *blocked* to *open* the classes that must be. Once L has been completely read, accept: as $|L| = |ID|$, all elements are now used.

It is clear by construction that, if this greedy algorithm accepts, then there is a linear extension of Γ' that realizes L with finishing order π; indeed, when the algorithm succeeds, then it has clearly respected the finishing order π, and whenever an identifier id of Γ' is marked as *used* by the algorithm, then id has the right value relative to the element of L that has just been read, and id is in an open class so no order relations of Γ' are violated by enumerating id at this point of the linear extension. The interesting direction is the converse: show that if L can be realized by a linear extension $<'$ of Γ' with finishing order π, then the algorithm accepts when considering π. To do so, we must show that if there is such a linear extension, then there is such a linear extension where identifiers are enumerated as in the greedy algorithm, i.e., we always choose an identifier with the right value and in the open class with the smallest finishing time: we call this a *minimal* identifier. (Note that we do not need to worry about which identifier is chosen: once we have decided on the value of the identifier and on its class, it does not matter which element we choose, because all elements in the class are unordered and have the same order relations to elements outside the class thanks to indistinguishability.) If we can prove this, then it justifies the existence of a linear extension that the greedy algorithm will construct, which we call a *greedy linear extension*.

Hence, let us see why it is always possible to enumerate minimal identifiers. Consider a linear extension $<'$ and take the smallest position in L where $<'$ chooses an identifier id which is non-minimal. We know that id must still have the correct value, i.e., $T(id)$ is determined, and by the definition of a linear extension, we know that id must be in an open class. Hence, we know that the class A of id is non-minimal, i.e., there is another open class A' containing an unused element with value $T(id)$, and A' is before A in the finishing order π. Let us take for A' the first open class with such an unused element in the finishing order π, and let id' be a minimal element, i.e., an element of A' with $T(id') = T(id)$. Let us now construct a different linear extension $<''$ by swapping id and id', i.e., enumerating id' instead of id, and enumerating id in $<''$ at the point where $<'$ enumerates id'. It is clear that the sequence of values (images by T) of the identifiers in $<''$ is still the same as in $<'$. Hence, if we can show that $<''$ additionally satisfies the order constraints of Γ', then we will have justified the existence of a linear extension that enumerates minimal identifiers until a later position; so, reapplying the rewriting argument, we will deduce the existence of a greedy linear extension. So it only remains to show that $<''$ satisfies the order constraints of Γ'.

Let us assume by way of contradiction that $<''$ violates an order constraint of Γ'. There are two possible kinds of violation. The first kind is if $<'$ enumerates an element id'' between id and id' for which $id < id''$, so that having $id'' <'' id$ in $<''$ is a violation. The second kind is if $<'$ enumerates an element id'' between

id and id' for which $id'' < id'$, so that having $id'' <'' id'$ in $<''$ is a violation. The second kind of violation cannot happen because id' is in an open class when $<'$ considers id, i.e., we have ensured that id' can be enumerated instead of id. Hence, we focus on violations of the first kind. Consider id'' such that $id <' id'' <' id'$ and let us show that $id \not< id''$. Letting A'' be the class of id'', we assume that $A'' \neq A$, as otherwise there is nothing to show because the classes are antichains. Now, we know from $<'$ that $id' \not<' id''$, and that the class A' of id' is not exhausted when $<'$ enumerates id''. As $<'$ respects the finishing order π, and A' comes before A in π, we know that A is not exhausted either when $<'$ enumerates id''. Letting id_A be an element of A which is still unused when $<'$ enumerates id'', we know that $id_A \not< id''$. So, as $id'' \notin A$, by indistinguishability, we have $id \not< id''$. This is what we wanted to show, so id'' cannot witness a violation of the first kind. Hence $<''$ does not violate the order constraints of Γ', and repeating this rewriting argument shows that there is a greedy linear extension that the greedy algorithm will find, contradicting our assumption. This establishes our result in the case where we only have the bounded-ia-width po-relation Γ'.

We now return to the general case where the bounded-width po-relation Γ is not empty. In this case, we will again enumerate all possible finishing orders for the classes of Γ', of which there are constantly many, and apply an algorithm for each finishing order π, with the algorithm succeeding iff it succeeds for some finishing order.

We first observe that if there is a way to achieve L as a possible world of $\Gamma \cup \Gamma'$ for a finishing order π, then there is one where the subsequence of the tuples that are matched to Γ' are matched following the greedy strategy as we presented before. This is simply because L must then be an interleaving of a possible world of Γ and a possible world of Γ', and a match for the possible world of Γ' can be found as a greedy match, by what was shown above. So it suffices to assume that the tuples matched to Γ' are matched following the greedy algorithm that we previously described.

Second, we observe the following: for any prefix L' of L and order ideal Γ'' of Γ, if we realize L' by matching exactly the tuples of Γ'' in Γ, and by matching the other tuples to Γ' following the greedy algorithm, then the matched tuples in Γ' are entirely determined (up to replacing tuples in a class by other tuples with the same value). This is because, while there may be multiple ways to match parts of L' to Γ'' in a way that leaves a different sequence of tuples to be matched to Γ', all these ways make us match the same bag of tuples to Γ'; now the state of Γ' after matching a bag of tuples following the greedy algorithm (for a fixed finishing order) is the same, no matter the order in which these tuples are matched, assuming that the match does not fail.

This justifies that we can solve the problem with a dynamic algorithm again. The state contains the position $\mathbf{m}$ in each chain of Γ, and a position i in the candidate possible world. As in the proof of Lemma 21, we filter the configurations so that they are sane with respect to the order constraints between the chains of Γ. For each state, we will store a Boolean value indicating whether the prefix of length i of L can be realized by $\Gamma \cup \Gamma'$ such that the tuples of Γ

that are matched is the order ideal $s(\mathbf{m})$ described by $\mathbf{m}$, and such that the other tuples of the prefix are matched to Γ' following the greedy algorithm with finishing order π. By our second remark above, when the Boolean is true, the state of Γ' is uniquely determined, and we also store it as part of the state (it is polynomial) so that we do not have to recompute it each time.

From each state we can make progress by consuming the next tuple from the candidate possible world, increasing the length of the prefix, and reaching one of the following states: either match the tuple to a chain of Γ, in which case we make progress in one chain and the consumed tuples in Γ' remain the same; or make progress in Γ', in which case we look at the previous state of Γ' that was stored and consume a tuple from Γ' following the greedy algorithm: more specifically, we find an unused tuple with the right label which is in the open class that appears first in the finishing order, if the class is now exhausted we verify that it was supposed to be the next one according to the finishing order, and we update the open, exhausted and blocked status of the classes.

Applying the dynamic algorithm allows us to conclude whether L can be realized by matching all tuples of Γ, and matching tuples in Γ' following the greedy algorithm with finishing order π (and checking cardinality suffices to ensure that we have matched all tuples of Γ'). If the answer of the dynamic algorithm is YES, then it is clear that, following the path from the initial to the final state found by the dynamic algorithm, we can realize L. Conversely, if L can be realized, then by our preliminary remark it can be realized in a way that matches tuples in Γ' following the greedy algorithm for some finishing order. Now, for that finishing order, the path of the dynamic algorithm that matches tuples to Γ or to Γ' following that match will answer YES. □

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 inputs.

Theorem 31. *There is a PosRA$_{LEX}$ query and a PosRA$_{DIR}$ 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.*

Proof. The proof is by adapting the proof of Theorem 22. The argument is exactly the same, except that we take relation S to be *unordered* rather than totally ordered. Intuitively, in Figure 4, this means that we drop the vertical edges in the grid. The proof adapts, because it only used the fact that $id'_j < id'_k$ for $j < k$ within a row-i; we never used the comparability relations across rows. □

6. Tractable Cases for Accumulation Queries

We next study POSS and CERT *in presence of accumulation*. Recall that in the general case, POSS is NP-hard and CERT is coNP-hard, so we study tractable cases in this section.

6.1. Cancellative Accumulation

We first study the case where accumulation is performed in a *cancellative* monoid (recall Definition 11). This large class of accumulation functions includes the top-k operator (defined above Example 16) and both operators in Example 13. We design an efficient algorithm for certainty in this case.

Theorem 32. *CERT is in PTIME for any fixed $PosRA^{\text{acc}}$ query that performs accumulation in a cancellative monoid.*

To prove this result, we define a notion of *possible ranks* for pairs of incomparable elements, and define a *safe swaps* property, intuitively designed to ensure that we have only one possible world.

Definition 33. *Let $P = (ID, <)$ be a poset. For $x \in ID$, we call $A_x := \{y \in ID \mid y < x\}$ the* ancestors *of x and call $D_x := \{y \in ID \mid x < y\}$ the* descendants *of x.*

Now, given two incomparable *elements x and y in ID, we define the* possible ranks $\mathrm{pr}_P(x, y)$ *as the interval $[a + 1, |ID| - d]$, where $a := |A_x \cup A_y|$ and $d := |D_x \cup D_y|$.*

Let $(\mathcal{M}, \oplus, \varepsilon)$ be a monoid and let $h : \mathcal{D} \times \mathbb{N} \to \mathcal{M}$ be an accumulation map. Let Γ be a po-relation with underlying poset P. We say that Γ has the safe swaps *property with respect to $\oplus$ and h if the following holds: for any pair $x \neq y$ of incomparable identifiers of Γ, for any pair $p, p+1$ in $\mathrm{pr}_P(x, y)$, we have*

$$h(T(x), p) \oplus h(T(y), p + 1) = h(T(y), p) \oplus h(T(x), p + 1).$$

We first show the following soundness result for possible ranks.

Lemma 34. *For any poset $P = (ID, <)$ and incomparable elements $x, y \in ID$, for any $p \neq q \in \mathrm{pr}_P(x, y)$, we can compute in PTIME a linear extension $<'$ of P in which element x is enumerated at position p, and element y is enumerated at position q.*

Proof. We write $a := |A_x \cup A_y|$ and $d := |D_x \cup D_y|$. We will build the desired linear extension $<'$ by enumerating all elements of $A_x \cup A_y$ in any order at the beginning, and enumerating all elements of $D_x \cup D_y$ at the end: this can be done without enumerating either x or y because x and y are incomparable.

Let $p' := p - a$, and $q' := q - a$; it follows from the definition of $\mathrm{pr}_P(x, y)$ that $1 \leqslant p', q' \leqslant |ID| - d - a$, and clearly $p' \neq q'$.

Now, all elements that are not enumerated by $<'$ are either x, y, or incomparable to both x and y. Consider any linear extension $<''$ of these unenumerated elements except x and y; it has length $|ID| - d - a - 2$. Now, as $p' \neq q'$, if $p' < q'$, then we can enumerate $p' - 1$ of these elements, enumerate x, enumerate $q' - p' - 1$ of these elements, enumerate y, and enumerate the remaining elements, following $<''$. We proceed similarly, reversing the roles of x and y, if $q' < p'$. We have constructed $<'$ in PTIME and it clearly has the required properties. □

We can then show that the safe swaps criterion is tractable to verify.

Lemma 35. *For any fixed (PTIME-evaluable) accumulation operator* $\mathrm{accum}_{h,\oplus}$ *we can determine in PTIME, given a po-relation* Γ*, whether* Γ *has safe swaps with respect to* $\oplus$ *and* h*.*

Proof. Consider each pair (id_1, id_2) of elements of Γ and check in PTIME whether they are incomparable. If this is the case, compute in PTIME $\mathrm{pr}_\Gamma(id_1, id_2)$ and for each pair p, $p{+}1$ of consecutive integers, compute $h(T(id_1), p) \oplus h(T(id_2), p{+}1)$ and $h(T(id_2), p) \oplus h(T(id_1), p+1)$ in PTIME (this uses PTIME-evaluability of the accumulation operator), and check whether they are equal. □

We last show the following lemma, from which we will easily be able to prove Theorem 32.

Lemma 36. *For any (PTIME-evaluable) accumulation operator* $\mathrm{accum}_{h,\oplus}$ *on a cancellative monoid* $(\mathcal{M}, \oplus, \varepsilon)$*, for any po-relation* Γ*, we have* $|\mathrm{accum}_{h,\oplus}(\Gamma)| = 1$ *iff* Γ *has safe swaps with respect to* $\oplus$ *and* h*.*

Proof. For the forward direction, assume that Γ does *not* have the safe swaps property. Hence, there exist two incomparable identifiers id_1 and id_2 in Γ and a pair of consecutive integers $p, p+1$ in $\mathrm{pr}_\Gamma(id_1, id_2)$ such that:

$$h(T(id_1), p) \oplus h(T(id_2), p+1) \neq h(T(id_2), p) \oplus h(T(id_1), p+1) \qquad (1)$$

We use Lemma 34 to compute two possible worlds L and L' of Γ, where id_1 and id_2 occur respectively at positions p and $p+1$ in L, and at positions $p+1$ and p respectively in L': from the proof of Lemma 34 it is clear that we can ensure that L and L' are otherwise identical. As accumulation is associative, we know that $\mathrm{accum}_{h,\oplus}(\Gamma) = v \oplus h(T(id_1), p) \oplus h(T(id_2), p+1) \oplus v'$, where v is the result of accumulation on the tuples in L before id_1, and v' is the result of accumulation on the tuples in L after id_2. Likewise, $\mathrm{accum}_{h,\oplus}(\Gamma) = v \oplus h(T(id_2), p) \oplus h(T(id_1), p+1) \oplus v'$. We then use cancellativity of $\mathcal{M}$ to deduce that these two values are different thanks to Equation (1). Hence, L and L' are possible worlds of Γ that yield different accumulation results, so we conclude that $|\mathrm{accum}_{h,\oplus}(\Gamma)| > 1$.

For the backward direction, assume that Γ has the safe swaps property. Assume by way of contradiction that there are two possible worlds $L_1, L_2 \in pw(\Gamma)$ such that $w_1 := \mathrm{accum}_{h,\oplus}(L_1)$ and $w_2 := \mathrm{accum}_{h,\oplus}(L_2)$ are different. Take L_1 and L_2 to have the longest possible common prefix, i.e., the first position i such that L_1 and L_2 enumerate a different identifier at position i is as large as possible. Let $0 \leqslant i_0 < |\Gamma|$ be the length of the common prefix. Let Γ' be the result of removing from Γ the identifiers enumerated in the common prefix of L_1 and L_2, and let L_1' and L_2' be L_1 and L_2 without their common prefix. Let $id_1 \neq id_2$ be the first identifiers enumerated by L_1' and L_2'; it is immediate that id_1 and id_2 are roots of the underlying poset of Γ', that is, no element of Γ' is less than them. Further, it is clear that the result w_1' of performing accumulation over L_2' (but offsetting all ranks by i_0), and the result w_2' of performing accumulation over L_1' (also offsetting all ranks by i_0), are different. Indeed, by the contrapositive of

cancellativity, combining w'_1 and w'_2 with the accumulation result of the common prefix leads to the different accumulation results w_1 and w_2.

Our goal is to construct a possible world $L'_3 \in pw(\Gamma')$ which starts by enumerating id_1 but ensures that the result of accumulation on L'_3 (again offsetting all ranks by i_0) is w'_2. If we can build such a possible world L'_3, then combining it with the common prefix will give a possible world L_3 of Γ such that the result of accumulation on L_3 is $w_2 \neq w_1$, yet L_1 and L_3 have a common prefix of length $> i_0$, contradicting minimality. Hence, it suffices to show how to construct such a possible world L'_3.

As id_1 is an identifier of Γ', there must be a position where L'_2 enumerates id_1, and all identifiers before id_1 in L'_2 cannot be descendants of id_1: as id_1 is a root of Γ', these identifiers must be incomparable to id_1. Write the sequence of these identifiers in L'_2 as $L''_2 = id'_1, \ldots, id'_m$, and let L'''_2 be the sequence following id_1, so that L'_2 is the concatenation of L''_2, id_1, and L'''_2. We now consider the following sequence of list relations, which are clearly possible worlds of Γ', where we intuitively move id_1 to the beginning of the list via successive swaps:

$$
\begin{aligned}
&id'_1 \ldots id'_{m-2}\ id'_{m-1}\ id'_m\ \underline{id_1}\ L'''_2,\\
&\quad id'_1 \ldots id'_{m-2}\ id'_{m-1}\ \underline{id_1}\ id'_m\ L'''_2,\\
&\qquad id'_1 \ldots id'_{m-2}\ \underline{id_1}\ id'_{m-1}\ id'_m\ L'''_2,\\
&\qquad\qquad \vdots\\
&\qquad id'_1\ id'_2\ \underline{id_1}\ id'_3 \ldots id'_{m-2}\ id'_{m-1}\ id'_m\ L'''_2,\\
&\qquad\quad id'_1\ \underline{id_1}\ id'_2\ id'_3 \ldots id'_{m-2}\ id'_{m-1}\ id'_m\ L'''_2,\\
&\qquad\qquad \underline{id_1}\ id'_1\ id'_2\ id'_3 \ldots id'_{m-2}\ id'_{m-1}\ id'_m\ L'''_2.
\end{aligned}
$$

We can see that any consecutive pair in this list achieves the same accumulation result. To do so, consider any pair of consecutive lists in this sequence, and observe that the two lists only differ at two successive identifiers, i.e., the first list contains $id'_j id_1$ and the second contains $id_1 id'_j$ for some $1 \leqslant j \leqslant m$. Thus, it suffices to show that the accumulation result for $id'_j id_1$ and $id_1 id'_j$ is the same, and this is exactly what the safe swaps property for id_1 and id'_j says, as it is easily checked that $j, j+1 \in \mathrm{pr}_{\Gamma'}(id'_j, id_1)$, so that $j+i_0, j+i_0+1 \in \mathrm{pr}_{\Gamma}(id'_j, id_1)$. Now, the first list relation above is L'_2, and the last list relation above starts by id_1, so we have built our desired L'_3. This establishes the second direction of the proof and concludes. □

We are now ready to prove Theorem 32.

Proof. Given the instance (D, v) of the `CERT` problem for the query Q with accumulation operator $\mathrm{accum}_{h,\oplus}$, we use Proposition 2 to build $\Gamma := Q(D)$ in PTIME. We then use Lemma 35 to test in PTIME whether Γ has safe swaps with respect to $\oplus$ and h. If it does not, then, by Lemma 36, v cannot be certain, so (D, v) is not a positive instance of `CERT`. If it does, then, by Lemma 36, $Q(D)$

has only one possible world, so we can compute an arbitrary linear extension of Γ, obtain one possible world $L \in pw(\Gamma)$, check whether $\mathrm{accum}_{h,\oplus}(L) = v$, and decide CERT accordingly. □

We have shown Theorem 32 on PosRA$^{\mathrm{acc}}$ queries. Note that this result clearly implies that CERT is also tractable for PosRA queries, as we claimed in Section 4: indeed, we can translate any PosRA query to a PosRA$^{\mathrm{acc}}$ query that uses a dummy accumulation operator in the concatenation monoid, and hence the CERT problem for PosRA queries reduces to the CERT problem for PosRA$^{\mathrm{acc}}$ queries in this fixed cancellative monoid. The same reasoning applied to Theorem 22 implies that the POSS problem for PosRA$^{\mathrm{acc}}$ is NP-hard even on cancellative monoids, in contrast with Theorem 32.

6.2. Finite and Position-Invariant Accumulation

We have shown that CERT (but not POSS) is tractable on cancellative accumulation operators. It is then natural to wonder whether a similar result holds when assuming that accumulation is finite and position-invariant (recall Definition 11). We will now show that these restrictions do *not* suffice to make POSS and CERT tractable. However, we will show in Section 6.3 that they can ensure tractability when we combine them with assumptions on the input po-relations.

We start by showing that POSS is intractable.

Theorem 37. *There is a PosRA$^{\mathrm{acc}}$ query with a finite and position-invariant accumulation operator for which POSS is NP-hard even assuming that the input po-database contains only totally ordered po-relations.*

To prove this result, we define the following finite domains:

- $\mathcal{D}_- := \{\mathsf{s}_-, \mathsf{n}_-, \mathsf{e}_-\}$ (the element names used here intuitively correspond to the names used in the proof of Theorem 22);
- $\mathcal{D}_+ := \{\mathsf{s}_+, \mathsf{n}_+, \mathsf{e}_+\}$;
- $\mathcal{D}_\pm := \mathcal{D}_- \sqcup \mathcal{D}_+ \sqcup \{\mathsf{l}, \mathsf{r}\}$ (the additional elements stand for "left" and "right").

We define the following regular expression on $\mathcal{D}_\pm^*$, and call *balanced* a word that satisfies it:

$$e := \mathsf{l}\,(\mathsf{s}_-\mathsf{s}_+|\mathsf{n}_-\mathsf{n}_+|\mathsf{e}_-\mathsf{e}_+)^*\,\mathsf{r}$$

We now define the following problem.

Definition 38. *The* balanced checking problem *for a PosRA query Q asks, given a po-database D of po-relations over $\mathcal{D}_\pm$, whether there is $L \in pw(Q(D))$ such that L is balanced, i.e., it has arity 1, its domain is $\mathcal{D}_\pm$, and L satisfies e when seen as a word over $\mathcal{D}_\pm$.*

We also introduce the following regular expression: $e' := \mathsf{l}\,\mathcal{D}_\pm^*\,\mathsf{r}$, which we will use later to guarantee that there are only two possible worlds. We now show that the balanced checking problem is intractable.

Lemma 39. *There exists a PosRA query Q_b over po-databases with domain in $\mathcal{D}_\pm$ such that the balanced checking problem for Q_b is NP-hard, even when all input po-relations are totally ordered. Further, Q_b is such that, for any input po-database D, all possible worlds of $Q_\mathsf{b}(D)$ satisfy e'.*

To prove this lemma, recall the definition of $\cup_{\mathtt{CAT}}$ (Definition 5), and recall from Lemma 6 that $\cup_{\mathtt{CAT}}$ can be expressed by a PosRA query. We construct the query $Q'_\mathsf{b}(R,T) := [\mathsf{l}] \cup_{\mathtt{CAT}} ((R \cup T) \cup_{\mathtt{CAT}} [\mathsf{r}])$, i.e., the union of R and T, preceded by l and followed by r.

For any word $w \in \mathcal{D}_+^*$, we denote by L_w^+ the unary list relation defined by mapping each letter of w to the corresponding letter in $\mathcal{D}_+$, we define L_w^- analogously for $\mathcal{D}_-$, and we write Γ_w^- for the totally ordered po-relation with $pw(\Gamma_w^-) = \{L_w^-\}$. We now claim that the balanced checking problem for Q'_b can be rephrased in terms of the possibility problem.

Lemma 40. *For any $w \in \mathcal{D}_+^*$ and unary po-relation Γ over $\mathcal{D}_+$, we have $L_w^+ \in pw(\Gamma)$ iff the po-database D mapping R to Γ_w^- and T to Γ is a positive instance to the balanced checking problem for Q'_b.*

Proof. For the forward direction, assume that w is indeed a possible world L of Γ and let us construct a balanced possible world L' of $Q'_\mathsf{b}(D)$. L' starts with l. Then, L' alternatively enumerates one tuple from Γ_w^- (in their total order) and one from Γ (taken in the order of the linear extension that yields L). Finally, L' ends with r. It is clear that L' is balanced.

For the backward direction, observe that a balanced possible world of $Q'_\mathsf{b}(D)$ must start by l, finish by r, and, between the two, it must alternatively enumerate tuples from Γ_w^- in their total order and tuples from one of the possible worlds $L \in pw(\Gamma)$: it is clear that L then achieves w. □

We now use Lemma 40 to prove Lemma 39.

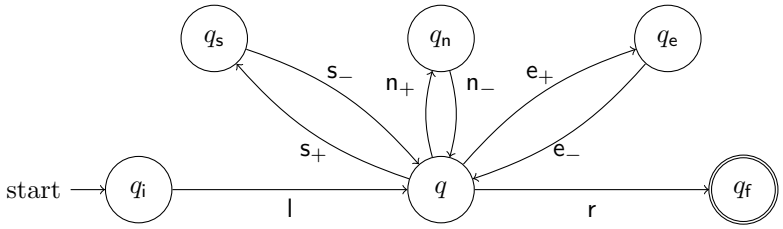
Proof. By Theorem 22, there is a query Q_0 in PosRA such that the POSS problem for Q_0 is NP-hard, even for totally ordered input relations. What is more, by inspecting the construction in the proof of Theorem 22, we can observe that the output arity of Q_0 is 1, and that the input relations can be assumed to have domain $\mathcal{D}_+$: indeed, the input po-relation S defined as $[\leqslant 3m-1]$ uses labels that are irrelevant (they are projected away), and the input po-relation S' uses only labels from $\{\mathsf{s}, \mathsf{n}, \mathsf{e}\}$, so we can rename them to $\{\mathsf{s}_+, \mathsf{n}_+, \mathsf{e}_+\}$. We now define the PosRA query Q_b: its input relations are those of Q_0 plus a fresh relation name R, and it maps any po-relation Γ' for R and input po-database D for Q_0 to $Q'_\mathsf{b}(\Gamma', Q_0(D))$. By definition of Q'_b, our query Q_b clearly satisfies the additional condition that all possible worlds satisfy e'.

We reduce the POSS problem for Q_0 to the balanced checking problem for Q_b in PTIME. More specifically, we claim that (D, L) is a positive instance to POSS for Q_0 iff D' is a positive instance to the balanced checking problem for Q_b, where D' is obtained from D by adding the totally ordered relation Γ_w^- to interpret the fresh name R, with w the word on $\mathcal{D}_+$ achieved by L. But this is exactly what Lemma 40 shows, for $\Gamma := Q_0(D)$. This concludes the reduction,

so we have shown that the balanced checking problem for Q_{b} is NP-hard, even assuming that the input po-database (here, D') contains only totally ordered po-relations. □

To prove our hardness result for POSS (Theorem 37), we will now reduce the balanced checking problem to POSS, using an accumulation operator to do the job. We will further ensure that there are at most two possible results, which will be useful for CERT later. To do this, we need to introduce some new concepts.

We define a deterministic complete finite automaton A as follows, where all omitted transitions go to a sink state $q_\bot$ not shown in the picture. It is clear that A recognizes the language of the regular expression e.



We let S be the state space of A, and use it to define the *transition monoid* of A, which is a finite monoid (so we are indeed performing finite accumulation). Let $\mathcal{F}_S$ be the finite set of total functions from S to S, and consider the monoid defined on $\mathcal{F}_S$ with the identity function id as the neutral element, and with function composition $\circ$ as the (associative) binary operation. We define inductively a mapping h from $\mathcal{D}_\pm^*$ to $\mathcal{F}_S$ as follows, which can be understood as a homomorphism from the free monoid $\mathcal{D}_\pm^*$ to the transition monoid of A:

- For ε the empty word, $h(\varepsilon)$ is the identity function id.
- For $a \in \mathcal{D}_\pm$, $h(a)$ is the transition table for symbol a for the automaton A, i.e., the function that maps each state $q \in S$ to the one state q' such that there is an a-labeled transition from q to q'; the fact that A is deterministic and complete is what ensures that this is well-defined.
- For $w \in \mathcal{D}_\pm^*$ and $w \neq \varepsilon$, writing $w = aw'$ with $a \in \mathcal{D}_\pm$, we define $h(w) := h(w') \circ h(a)$.

It is easy to show inductively that, for any $w \in \mathcal{D}_\pm^*$, and for any $q \in S$, the state $(h(w))(q)$ is the one that we reach in A when reading the word w from the state q. We will identify two special elements of $\mathcal{F}_S$:

- f_0, the function mapping every state of S to the sink state $q_\bot$;
- f_1, the function mapping the initial state q_{i} to the final state q_{f}, and mapping every other state in $S \backslash \{q_{\mathsf{i}}\}$ to $q_\bot$.

Recall the definition of the regular expression e' earlier. We claim the following property on the automaton A.

Lemma 41. *For any word $w \in \mathcal{D}^*_{\pm}$ that matches e', we have $h(w) = f_1$ if w is balanced (i.e., satisfies e) and $h(w) = f_0$ otherwise.*

Proof. By the definition of A, for any state $q \neq q_{\mathsf{i}}$, we have $(h(\mathsf{l}))(q) = q_\perp$, so that, as $q_\perp$ is a sink state, we have $(h(w))(q) = q_\perp$ for any w that satisfies e'. Further, by definition of A, for any state q, we have $(h(\mathsf{r}))(q) \in \{q_\perp, q_{\mathsf{f}}\}$, so that, for any state q and w that satisfies e', we have $(h(w))(q) \in \{q_\perp, q_{\mathsf{f}}\}$. This implies that, for any word w that satisfies e', we have $h(w) \in \{f_0, f_1\}$.

Now, as we know that A recognizes the language of e, we have the desired property, because, for any w satisfying e', $h(w)(q_{\mathsf{i}})$ is q_{f} or not depending on whether w satisfies e or not, so $h(w)$ is f_1 or f_0 depending on whether w satisfies e or not. □

This ensures that we have only two possible accumulation results, and that they accurately test whether the input word is balanced. We can now prove our hardness result for POSS, Theorem 37.

Proof. Consider the query Q_{b} whose existence is guaranteed by Lemma 39, and remember that all its possible worlds on any input po-database must satisfy e'. Construct now the query $Q_{\mathsf{a}} := \mathrm{accum}_{h,\circ}(Q_{\mathsf{b}})$, using the mapping h that we defined above, seen as a position-invariant accumulation map. We conclude the proof by showing that POSS is NP-hard for Q_{a}, even when the input po-database consists only of totally ordered po-relations. To see that this is the case, we reduce the balanced checking problem for Q_{b} to POSS for Q_{a} with the trivial reduction: we claim that for any po-database D, there is a balanced possible world in $Q_{\mathsf{b}}(D)$ iff $f_1 \in Q_{\mathsf{a}}(D)$, which is proved by Lemma 41. Hence, $Q_{\mathsf{b}}(D)$ is balanced iff (D, f_1) is a positive instance of POSS for Q_{a}. This concludes the reduction, and establishes our hardness result. □

We last show an analogue of Theorem 37 for CERT as well.

Theorem 42. *There is a PosRA$^{\mathrm{acc}}$ query with a finite and position-invariant accumulation operator for which CERT is coNP-hard even assuming that the input po-database contains only totally ordered po-relations.*

Proof. Consider the query Q_{a} from Theorem 37. We show a PTIME reduction from the NP-hard problem of POSS for Q_{a} (for totally ordered input po-databases) to the negation of the CERT problem for Q_{a} (for input po-databases of the same kind).

Consider an instance of POSS for Q_{a} consisting of an input po-database D and candidate result $v \in \mathcal{M}$. Recall that the query Q_{a} uses accumulation, so it is of the form $\mathrm{accum}_{h,\oplus}(Q')$. Evaluate $\Gamma := Q'(D)$ in PTIME by Proposition 2, and compute in PTIME an arbitrary possible world $L' \in pw(\Gamma)$ by picking an arbitrary linear extension of Γ. Let $v' = \mathrm{accum}_{h,\oplus}(L')$. If $v = v'$ then (D, v) is a positive instance for POSS for Q_{a}. Otherwise, we have $v \neq v'$. Now, solve the CERT problem for Q_{a} on the input (D, v'). If the answer is YES, then (D, v) is a negative instance for POSS for Q_{a}. Otherwise, there must exist a possible world L'' in $pw(\Gamma)$ with $v'' = \mathrm{accum}_{h,\oplus}(L'')$ and $v'' \neq v'$. However, $|pw(Q_{\mathsf{a}}(D))| \leqslant 2$

and thus, as $v \neq v'$ and $v' \neq v''$, we must have $v = v''$. So (D, v) is a positive instance for POSS for $Q_{\mathtt{a}}$. This finishes the reduction and shows that CERT for $Q_{\mathtt{a}}$ is coNP-hard. □

6.3. Revisiting Section 5

We now know that finiteness and position-invariance do not suffice to ensure the tractability of POSS and CERT. In this section, we will show that they can nevertheless be used to obtain tractability when combined with assumptions on the input po-database, as we did in Section 5. Specifically, in the rest of this section, we will always assume that accumulation is finite, and we will sometimes assume that it is position-invariant. We call $\mathrm{PosRA}^{\mathrm{acc}}_{\mathtt{LEX}}$ and $\mathrm{PosRA}^{\mathrm{acc}}_{\mathrm{no}\times}$ the extension of $\mathrm{PosRA}_{\mathtt{LEX}}$ and $\mathrm{PosRA}_{\mathrm{no}\times}$ with accumulation.

We can first generalize our width-based tractability result on $\mathrm{PosRA}_{\mathtt{LEX}}$ (Theorem 20) to $\mathrm{PosRA}^{\mathrm{acc}}_{\mathtt{LEX}}$ queries with *finite* accumulation.

Theorem 43. *For any $\mathrm{PosRA}^{\mathrm{acc}}_{\mathtt{LEX}}$ query with a* finite *accumulation operator,* POSS *and* CERT *are in PTIME on po-databases of bounded width.*

To show this, as in Section 5, we can use Proposition 2 and Lemma 4 to argue that it suffices to show the following analogue of Lemma 21. Note that we compute exactly the (finite) set of all possible accumulation results, so this allows us to answer both POSS and CERT.

Lemma 44. *For any constant $k \in \mathbb{N}$, and finite accumulation operator $\mathrm{accum}_{h,\oplus}$, we can compute in PTIME, for any input po-relation Γ with width $\leqslant k$, the set $\mathrm{accum}_{h,\oplus}(\Gamma)$.*

Proof. We extend the proof of Lemma 21 and reuse its notation. For every sane vector $\mathbf{m}$, we now write $t(\mathbf{m}) := \mathrm{accum}_{h,\oplus}(T(s(\mathbf{m})))$, where $T(s(\mathbf{m}))$ denotes the sub-po-relation of Γ with the tuples of the order ideal $s(\mathbf{m})$. In other words, $t(\mathbf{m})$ is the set of possible accumulation results for the sub-po-relation on the order ideal $s(\mathbf{m})$: as the accumulation monoid is fixed on finite, the set has constant size. It is immediate that $t(0, \ldots, 0) = \{\varepsilon\}$, i.e., the only possible result is the neutral element of the accumulation monoid, and that $t(n_1, \ldots, n_{k'}) = \mathrm{accum}_{h,\oplus}(\Gamma)$ is our desired answer. Recall that e_i denotes the vector consisting of $n-1$ zeros and a 1 at position i, for $1 \leqslant i \leqslant k'$, and that "$-$" denotes the component-wise difference of vectors. We now observe that, for any sane vector $\mathbf{m}$, we have

$$t(\mathbf{m}) = \bigcup_{\substack{1 \leqslant i \leqslant k' \\ m_i > 0}} \left\{ v \oplus h\left(T(\Lambda_i[m_i]), \sum_{i'} m_{i'}\right) \;\middle|\; v \in t(\mathbf{m} - e_i) \right\}, \tag{2}$$

where we set $t(\mathbf{m}) := \emptyset$ whenever $\mathbf{m}$ is not sane. The correctness of Equation (2) is shown as in the proof of Lemma 21: any linear extension of $s(\mathbf{m})$ must end with one of the maximal elements of $s(\mathbf{m})$, which must be one of the $\Lambda_i[m_i]$ for $1 \leqslant i \leqslant m$ such that $m_i > 0$, and the preceding elements must be a linear extension of the ideal where this element was removed (which must be an ideal,

i.e., $\mathbf{m} - e_i$ must be sane). Conversely, any sequence constructed in this fashion is indeed a linear extension. Thus, the possible accumulation results are computed according to this characterization of the linear extensions. We store with each possible accumulation result a witnessing totally ordered relation from which it can be computed in PTIME, namely, the linear extension prefix considered in the previous reasoning, so that we can use the PTIME-evaluability of the underlying monoid to ensure that all computations of accumulation results can be performed in PTIME.

As in the proof of Lemma 21, Equation (2) allows us to compute $t(n_1, \ldots, n_{k'})$ in PTIME by a dynamic algorithm, which is the set $\mathrm{accum}_{h,\oplus}(\Gamma)$ that we wished to compute. This concludes the proof. □

Second, we can adapt the tractability result for queries without product (Theorem 24) when accumulation is *finite* and *position-invariant*.

Theorem 45. *For any $\mathrm{PosRA}^{\mathrm{acc}}_{\mathrm{no}\times}$ query with a* finite *and* position-invariant *accumulation operator,* `POSS` *and* `CERT` *are in PTIME on po-databases whose relations have either bounded width or bounded ia-width.*

To do so, again, it suffices to show the following analogue of Lemma 30 for finite and position-invariant accumulation.

Lemma 46. *For any constant $k \in \mathbb{N}$, and finite and position-invariant accumulation operator* $\mathrm{accum}_{h,\oplus}$*, we can compute in PTIME, for any input po-relation Γ with width $\leqslant k$ and input po-relation Γ' with ia-width $\leqslant k$, the set* $\mathrm{accum}_{h,\oplus}(\Gamma \cup \Gamma')$.

Proof. We use Dilworth's theorem to compute in PTIME a chain partition of Γ, and we use Proposition 27 to compute in PTIME an ia-partition $A_1 \sqcup \cdots \sqcup A_n$ of minimal cardinality of Γ', with $n \leqslant k$.

We then apply a dynamic algorithm whose state consists of the following:

- for each chain in the partition of Γ, the position in the chain;
- for each class A of the ia-partition of Γ', for each element m of the monoid, the number of identifiers id of A such that $h(T(id), 1) = m$ that have already been used.

There are polynomially many possible states; for the second bullet point, this uses the fact that the monoid is finite, so its size is constant because it is fixed as part of the query. Also note that we use the rank-invariance of h in the second bullet point.

The possible accumulation results for each of the possible states can then be computed by a dynamic algorithm. At each state, we can decide to make progress either in a chain of Γ (ensuring that the element that we enumerate has the right image by h, and that the new vector of positions of the chains is still sane, i.e., yields an order ideal of Γ) or in a class of Γ' (ensuring that this class is open, i.e., it has no ancestors in Γ' that were not enumerated yet, and that it contains an element which has the right image by h). This algorithm is correct

because there is a bijection between the ideals of $\Gamma \cup \Gamma'$ and the pairs of ideals of Γ and of ideals of Γ'. Now, the dynamic algorithm considers all ideals of Γ as in the proof of Lemma 44, and it clearly considers all possible ideals of Γ' except that we identify ideals that only differ by elements in the same class which are mapped to the same value by h (but this choice does not matter because the class is an antichain and these elements are indistinguishable outside the class).

As in the proof of Lemma 44, we can ensure that all accumulation operations are in PTIME, using PTIME-evaluability of the accumulation operator, up to the technicality of storing at each state, for each of the possible accumulation results, a witnessing totally ordered relation from which to compute it in PTIME. □

We note that the finiteness assumption is important, as the previous result does not hold otherwise. Specifically, there is an accumulation operator that is *position-invariant* but not *finite*, for which POSS is NP-hard even on unordered po-relations and with a trivial query.

Theorem 47. *There is a position-invariant accumulation operator* $\mathrm{accum}_{h,\oplus}$ *such that* POSS *is NP-hard for the* $\mathit{PosRA}^{\mathrm{acc}}_{\mathrm{no}\times}$ *query* $Q := \mathrm{accum}_{h,\oplus}(R)$*, even on input po-databases where* R *is interpreted as an unordered relation.*

Proof. We consider the NP-hard partition problem: given a multiset S of integers, decide whether it can be partitioned into two sets S_1 and S_2 that have the same sum. Let us reduce an instance of the partition problem with this restriction to an instance of the POSS problem, in PTIME.

Let $\mathcal{M}$ be the monoid generated by the functions $f : x \mapsto -x$ and $g_a : x \mapsto x + a$ for $a \in \mathbb{Z}$ under the function composition operation. We have $g_a \circ g_b = g_{a+b}$ for all $a, b \in \mathbb{N}$, $f \circ f = id$, and $f \circ g_a = g_{-a} \circ f$, so we actually have $\mathcal{D} = \{g_a \mid a \in \mathbb{Z}\} \sqcup \{f \circ g_a \mid a \in \mathbb{Z}\}$. Further, $\mathcal{M}$ is actually a group, as we can define $(g_a)^{-1} = g_{-a}$ and $(f \circ g_a)^{-1} = f \circ g_a$ for all $a \in \mathbb{Z}$.

We fix $\mathcal{D} = \mathbb{N} \sqcup \{-1\}$. We define the position-invariant accumulation map h as mapping -1 to f and $a \in \mathbb{N}$ to g_a. We encode the partition problem instance S in PTIME to an unordered po-relation Γ_S with a single attribute, that contains one tuple with value s for each $s \in S$, plus one tuple with value -1. Consider the POSS instance for the query $\mathrm{accum}_{h,+}(\Gamma)$, on the po-database D where the relation name R is interpreted as the po-relation Γ_S, and for the candidate result $v := f \in \mathcal{M}$.

We claim that this POSS instance is positive iff the partition problem has a solution. Indeed, if S has a partition, let $s = \sum_{i \in S_1} i = \sum_{i \in S_2} i$. Consider the total order on Γ_S which enumerates the tuples corresponding to the elements of S_1, then the tuple -1, then the tuples corresponding to the elements of S_2. The result of accumulation is then $g_s \circ f \circ g_s$, which is f.

Conversely, assume that the POSS problem has a solution. Consider a witness total order of Γ_S; it must a (possibly empty) sequence of tuples corresponding to a subset S_1 of S, then the tuple -1, then a (possibly empty) sequence corresponding to $S_2 \subseteq S$. Let s_1 and s_2 respectively be the sums of these subsets of S. The result of accumulation is then $g_{s_1} \circ f \circ g_{s_2}$, which simplifies to $g_{s_1 - s_2} \circ f$.

Hence, we have $s_1 = s_2$, so that S_1 and S_2 are a partition witnessing that S is a positive instance of the partition problem.

As the reduction is in PTIME, this concludes the proof. □

Finally, as explained above Example 16, we can use accumulation 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 48. *For any PosRA query Q, the following problems are in PTIME.*

- ***select-at-k:*** *Given a po-database D, tuple value t, and position $k \in \mathbb{N}$, determine 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, determine 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, determine whether it is possible/certain that the first occurrence of t_1 precedes all occurrences of t_2.*

Proof. To solve each problem, we first compute the po-relation $\Gamma := Q(D)$ in PTIME by Proposition 2. We then address each problem in turn.

First, we show tractability for **select-at-k**. Considering the po-relation $\Gamma = (ID, T, <)$, we can compute in PTIME, for every element $id \in ID$, its *earliest index* $\mathsf{i}^-(id)$, which is the number of ancestors of id by $<$ plus one, and its *latest index* $\mathsf{i}^+(id)$, which is the number of elements of Γ minus the number of descendants of id. It is easily seen that for any element $id \in ID$, there is a linear extension of Γ where id appears at position $\mathsf{i}^-(id)$ (by enumerating first exactly the ancestors of id), or at position $\mathsf{i}^+(id)$ (by enumerating first everything except the descendants of id), or in fact at any position of $[\mathsf{i}^-(id), \mathsf{i}^+(id)]$, the *interval* of id (this is by enumerating first the ancestors of id, and then as many elements as needed that are incomparable to id, along a linear extension of these elements). Hence, select-at-k possibility for tuple t and position k can be decided by checking, for each $id \in ID$ such that $T(id) = t$, whether $k \in [\mathsf{i}^-(id), \mathsf{i}^+(id)]$, and answering YES iff we can find such an id. For select-at-k certainty, we answer NO iff we can find an $id \in ID$ such that $k \in [\mathsf{i}^-(id), \mathsf{i}^+(id)]$ but we have $T(id) \neq t$.

Second, we show tractability for **top-k**. Considering the po-relation $\Gamma = (ID, T, <)$, we consider each sequence of k elements of Γ, of which there are at most $|ID|^k$, i.e., polynomially many, as k is fixed. To solve possibility for top-k, we consider each such sequence $id_1, \ldots, id_k$ such that $(T(id_1), \ldots, T(id_k))$ is equal to the candidate list relation L, and we check if this sequence is indeed a prefix of a linear extension of Γ, i.e., whether, for each $i \in \{1, \ldots, k\}$, for any

$id \in ID$ such that $id < id_i$, if $id_i \in \{id_1, \ldots, id_{i-1}\}$, which we can do in PTIME. We answer YES iff we can find such a sequence.

For certainty, we consider each sequence $id_1, \ldots, id_k$ such that we have $(T(id_1), \ldots, T(id_k)) \neq L$, and we check whether it is a prefix of a linear extension in the same way: we answer NO iff we can find such a sequence.

Third, we show tractability for **tuple-level comparison**. We are given the two tuple values t_1 and t_2, and we assume that both are in the image of T, as the tuple-level comparison problem is vacuous otherwise.

For possibility, given the two tuple values t_1 and t_2, we consider each $id \in ID$ such that $T(id) = t_1$, and for each of them, we construct $\Gamma_{id} := (ID, T, <_{id})$ where $<_{id}$ is the transitive closure of $< \cup \{(id, id') \mid id' \in ID, T(id') = t_2\}$. We answer YES iff one of the Γ_{id} is indeed a po-relation, i.e., if $<_{id}$ as defined does not contain a cycle. This is correct, because it is possible that the first occurrence of t_1 precedes all occurrences of t_2 iff there is some identifier id with tuple value t_1 that precedes all identifiers with tuple value t_2, i.e., iff one of the Γ_{id} has a linear extension.

For certainty, given t_1 and t_2, we answer the negation of possibility for t_2 and t_1. This is correct because certainty is false iff there is a linear extension of Γ where the first occurrence of t_1 does not precede all occurrences of t_2, i.e., iff there is a linear extension where the first occurrence of t_2 is not after an occurrence of t_1, i.e., iff some linear extension is such that the first occurrence of t_2 precedes all occurrences of t_1, i.e., iff possibility is true for t_2 and t_1. □

7. Extensions

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

7.1. Group-By

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

Definition 49. *Let $(\mathcal{M}, \oplus, \varepsilon)$ be a monoid and $h : \mathcal{D}^k \times \mathbb{N}_{>0} \to \mathcal{M}$ be an accumulation map, and let $\mathbf{A} = A_1, ..., A_n$ be a sequence of attributes: we call* $\mathrm{accumGroupBy}_{h,\oplus,\mathbf{A}}$ *an* accumulation operator with group-by. *Letting L be a list relation with compatible schema, we define* $\mathrm{accumGroupBy}_{h,\oplus,\mathbf{A}}(L)$ *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* $\mathrm{accum}_{h,\oplus}(\sigma_{A_1=t.A_1 \wedge \cdots \wedge A_n=t.A_n}(L))$ *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* $\{\mathrm{accumGroupBy}_{h,\oplus,\mathbf{A}}(L) \mid L \in pw(\Gamma)\}$.

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 $\mathrm{PosRA}^{\mathrm{accGBy}}$ 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 PosRA$^{\text{accGBy}}$ queries. Of course, whenever POSS and CERT are hard for some PosRA$^{\text{acc}}$ query Q on some kind of input po-relations, then there is a corresponding 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 PosRA$^{\text{accGBy}}$ is hard even on totally ordered po-relations and without the $\times_{\texttt{DIR}}$ operator. This result contrasts with the tractability of POSS for PosRA$_{\texttt{LEX}}$ queries (Theorem 20) and for PosRA$^{\text{acc}}_{\texttt{LEX}}$ queries with finite accumulation (Theorem 43) on totally ordered po-relations.

Theorem 50. *There is a PosRA$^{\text{accGBy}}$ query Q with finite and position-invariant accumulation, not using $\times_{\texttt{DIR}}$, such that POSS for Q is NP-hard even on totally ordered po-relations.*

Proof. Let Q be the query $\text{accumGroupBy}_{\oplus,h,\{1\}}(Q')$, where we define

$$Q' := \Pi_{3,4}(\sigma_{.1=.2}(R \times_{\texttt{LEX}} (S_1 \cup S_2 \cup S_3))).$$

In the accumulation operator, the accumulation map h maps each tuple t to its second component. Further, we define the finite monoid $\mathcal{M}$ to be the *syntactic monoid* [17] of the language defined by the regular expression $\mathsf{s}(\mathsf{l}_+\mathsf{l}_-|\mathsf{l}_-\mathsf{l}_+)^*\mathsf{e}$, where s (for "start"), l_- and l_+, and e (for "end") are fresh values from $\mathcal{D}$: this monoid ensures that, for any non-empty word w over the alphabet $\{\mathsf{s}, \mathsf{l}_-, \mathsf{l}_+, \mathsf{e}\}$ that starts with s and ends with e, the word w evaluates to ε in $\mathcal{M}$ iff w matches this regular expression.

We reduce from the NP-hard 3-SAT problem: we are given a conjunction of clauses $C_1, \ldots, C_n$, with each clause being a disjunction of three literals, namely, a variable or negated variable among $x_1, \ldots, x_m$, and we ask whether there is a valuation of the variables such that the clause is true. We fix an instance of this problem. We assume without loss of generality that the instance has been preprocessed to ensure that no clause contained two occurrences of the same variable, i.e., we remove duplicate literals in clauses, and we remove any clause that contains two occurrences of the same variable with different polarities (as the clause is then vacuous). We further assume that the instance has been preprocessed to ensure that each clause contains exactly 3 variables: we do so by introducing three fresh variables d_1, d_2, and d_3, by adding all possible clauses $\pm d_1 \vee \pm d_2 \vee \pm d_3$ on these variables except $\neg d_1 \vee \neg d_2 \vee \neg d_3$ (i.e., seven clauses), and by padding the other clauses to three literals by adding distinct disjuncts chosen from the $\neg d_i$. It is clear that this does not change the semantics of the instance: any satisfying assignment of the original instance yields a satisfying assignment of the rewritten instance by setting d_1, d_2, and d_3 to true, and conversely any satisfying assignment to the rewritten instance must set d_1, d_2, and d_3 to true (any other assignment will violate the clause where each d_i has the polarity which is the opposite of its value in the assignment), so the padding literals are never used to make a clause true.

We define the relation R to be $[\leqslant m + 3]$. The totally ordered relations S_1, S_2, and S_3 consist of $3m + 2n$ tuple values defined as follows.

- First, for the tuples with positions from 1 to m (the "opening gadget"):
  - The first component is 1 for all tuples in S_1 and 0 for all tuples in S_2 and S_3 (so they do not join with R);
  - The second component is i for the i-th tuple in S_1 (and irrelevant for tuples in S_2 and S_3);
  - The third component is s for all these tuples.

  The intuition for the opening gadget is that it ensures that accumulation in each of the m groups will start with the start value s, used to disambiguate the possible monoid values and ensure that there is exactly one correct value.

- For the tuples with positions from $m+1$ to $2m$ (the "variable choice" gadget):
  - The first component is 2 for all tuples in S_1 and S_2 and 0 for all tuples in S_3 (so they do not join with R);
  - The second component is i for the $(m+i)$-th tuple in S_1 and in S_2 (and irrelevant for S_3);
  - The third component is l_- for all tuples in S_1 and l_+ for all tuples in S_2 (and irrelevant for S_3).

  The intuition for the variable choice gadget is that, for each group, we have two incomparable elements, one labeled l_- and one labeled l_+. Hence, any linear extension must choose to enumerate one after the other, committing to a valuation of the variables in the 3-SAT instance; to achieve the candidate possible world, the linear extension will then have to continue enumerating the elements of this group in the correct order.

- For the tuples with positions from $2m+1$ to $2m+2n$ (the "clause check" gadget), for each $1 \leqslant j \leqslant n$, letting $j' := 2n + j + 1$, we describe tuples j' and $j'+1$ in S_1, S_2, S_3:
  - The first component is $j+2$;
  - The second component carries values in $\{a, b, c\}$, where we write clause C_j as $\pm x_a \vee \pm x_b \vee \pm x_c$. Specifically, the tuple $j'+1$ in relations S_1, S_2, and S_3 have values a, b, and c respectively; and the tuple j' in relations S_1, S_2, and S_3 have values c, a, b respectively.
  - The third component carries values in $\{\mathsf{l}_-, \mathsf{l}_+\}$. In relation S_1, we give value l_+ to tuple $j'+1$ and value l_- to tuple j' if the first variable of C_j is positive, and we do the reverse if it is negative. We do the same in relations S_2 and S_3 depending on the polarity of the second and third variables of C_j, respectively.

The intuition for the clause check gadget is that, for each $1 \leqslant j \leqslant n$, the tuples at levels j' and $j' + 1$ check that clause C_j is satisfied by the valuation chosen in the variable choice gadget. Specifically, if we consider the order constraints on the two elements from the same group (i.e., second component) which are implied by the order chosen for this variable in the variable choice gadget, the construction ensures that these order constraints plus the comparability relations of the chains imply a cycle (that is, an impossibility) iff the clause is violated by the chosen valuation.

- For the tuples with positions from $2m + 2n + 1$ to $3m + 2n$ (the "closing gadget"), the definition is like the opening gadget but replacing e by s, namely:
  - The first component is $n + 3$ for all tuples in S_1 and 0 for all tuples in S_2 and S_3 (which again do not join with R);
  - The second component is i for the i-th tuple in S_1;
  - The third component is e for all these tuples.

  The intuition for the closing gadget is that it ensures that accumulation in each group ends with value e.

We define the candidate possible world to consist of a list relation of n tuples; the i-th tuple carries value i as its first component (group identifier) and the acceptation value from the monoid $\mathcal{M}$ as its second component (accumulation value). The reduction that we described is clearly in PTIME, so all that remains is to show correctness of the reduction.

To do so, we first describe the result of evaluating $\Gamma := Q'(R, S_1, S_2, S_3)$ on the relations described above. Intuitively, it is just like $\Pi_{2,3}(\sigma_{.2 \neq \text{“0”}}(S_1 \cup S_2 \cup S_3))))$, but with the following additional comparability relations: all tuples in all chains whose first component carried a value i are less than all tuples in all chains whose first component carried a value $j > i$. In other words, we add comparability relations across chains as we move from one "first component" value to the next. The point of this is that it forces us to enumerate the tuples of the chains in a way that "synchronizes" across all chains whenever we change the first component value. Observe that, in keeping with Lemma 4, the width of Γ has a constant bound, namely, 3.

Let us now show the correctness of the reduction. For the forward direction, consider a valuation ν that satisfies the 3-SAT instance. Construct the linear extension of Γ as follows.

- For the opening gadget, enumerate all tuples of S_1 in the prescribed order. Hence, the current accumulation result in all m groups is s.
- For the variable choice gadget, for all i, enumerate the i-th tuples of S_1 and S_2 of the gadget in an order depending on $\nu(x_i)$: if $\nu(x_i)$ is 1, enumerate first the tuple of S_1 and then the tuple of S_2, and do the converse if $\nu(x_i) = 0$. Hence, for all $1 \leqslant i \leqslant m$, the current accumulation result in group i is $\mathsf{s}\mathsf{l}_\mathsf{l}_+$ if $\nu(x_i)$ is 1 and $\mathsf{s}\mathsf{l}_+\mathsf{l}_$ otherwise.

- For the clause check gadget, we consider each clause in order, for $1 \leqslant j \leqslant n$, maintaining the property that, for each group $1 \leqslant i \leqslant n$, the current accumulation result in group i is of the form $\mathsf{s}(\mathsf{l_l_+})^*$ if $\nu(x_i) = 1$ and $\mathsf{s}(\mathsf{l_+l_})^*$ otherwise.

  Fix a clause C_j, let $j' := 2n + j + 1$ as before, and study the tuples j' and $j' + 1$ of S_1, S_2, S_3. As C_j is satisfied under ν, let x_d be the witnessing literal (with $d \in \{a, b, c\}$), and let d' be the index (in $\{1, 2, 3\}$) of variable d. Assume that x_d occurs positively; the argument is symmetric if it occurs negatively. By definition, $\nu(x_d) = 1$, and by construction tuple j' in relation $S_{1+(d'+1 \bmod 3)}$ carries value $\mathsf{l_}$ and it is in group d. Hence, we can enumerate it and group d now carries a value of the form $\mathsf{s}(\mathsf{l_l_+})^*\mathsf{l_}$. Now, letting x_e be the $1 + (d' + 1 \bmod 3)$-th variable of $\{x_a, x_b, x_c\}$, the two elements of group e (tuple $j' + 1$ of $S_{1+(d'+1 \bmod 3)}$ and tuple j' of $S_{1+(d'+1 \bmod 3)}$) both had all their predecessors enumerated; so we can enumerate them in the order that we prefer to satisfy the condition on the accumulation values; then we enumerate likewise the two elements in the remaining group in the order that we prefer, and last we enumerate the second element of group d; so we have satisfied the invariants.

- Last, for the closing gadget, we enumerate all tuples of S_1 and we have indeed obtained the desired accumulation result.

This concludes the proof of the forward direction.

For the backward direction, consider any linear extension of Γ. Thanks to the order constraints of Γ, the linear extension must enumerate tuples in the following order.

- First, all tuples of the opening gadget.

- Then, all tuples of the variable choice gadget. We use this to define a valuation ν: for each variable x_i, we set $\nu(x_i) = 1$ if the tuple of S_1 in group i was enumerated before the one in group S_2, and we set $\nu(x_i) = 0$ otherwise.

- Then, for each $1 \leqslant j \leqslant n$, in order, tuples $2n + j + 1$ of S_1, S_2, S_3.

  Observe that, for each value of j, just before we enumerate these tuples, it must be the case that the current accumulation value for every variable x_i is of the form $\mathsf{s}(\mathsf{l_l_+})^*$ if $\nu(x_i) = 1$, and $\mathsf{s}(\mathsf{l_+l_})^*$ otherwise. Indeed, fixing $1 \leqslant i \leqslant n$, assume the case where $\nu(x_i) = 1$ (the case where $\nu(x_i) = 0$ is symmetric). In this case, the accumulation state for x_i after the variable choice gadget was $\mathsf{sl_l_+}$, and each pair of levels in the clause check gadget made us enumerate either ε (variable x_i did not occur in the clause) or one of $\mathsf{l_l_+}$ or $\mathsf{l_+l_}$ (variable x_i occurred in the clause); as the 3-SAT instance was preprocessed to ensure that each variable occurred only at most once in each clause, this case enumeration is exhaustive. Hence, the only way to obtain the correct accumulation result is to always enumerate $\mathsf{l_l_+}$, as

if we ever do the contrary the accumulation result can never satisfy the regular expression that it should satisfy.

- Last, all tuples of the closing gadget.

What we have to show is that the valuation ν thus defined indeed satisfies the formula of the 3-SAT instance. Indeed, fix $1 \leqslant j \leqslant n$ and consider clause C_j. Let S_i be the first relation where the linear extension enumerated a tuple for the clause check gadget of C_j, and let x_d be its variable (where d is its group index). If $\nu(x_d) = 1$, then the observation above implies that the label of the enumerated element must be $\mathsf{I}_$, as otherwise the accumulation result cannot be correct. Hence, by construction, it means that variable x_d must occur positively in C_j, so x_d witnesses that ν satisfies C_j. If $\nu(x_d) = 0$, the reasoning is symmetric. This concludes the proof in the backwards direction, so we have established correctness of the reduction, which concludes the proof. □

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, and all CERT tractability results from Section 6 extend.

Theorem 51. *Theorems 32, 43, and 45 extend to the $\text{PosRA}^{\text{accGBy}}$ problem when imposing the same restrictions on query operators, accumulation, and input po-relations. Specifically:*

- *CERT is in PTIME for any fixed $\text{PosRA}^{\text{accGBy}}$ query that performs accumulation in a cancellative monoid.*
- *For any $\text{PosRA}^{\text{accGBy}}$ query not using the $\times_{DIR}$ operator and with a* finite *accumulation operator, POSS and CERT are in PTIME on po-databases of bounded width.*
- *For any $\text{PosRA}^{\text{accGBy}}$ query not using any product operator and with a* finite *and* position-invariant *accumulation operator, POSS and CERT are in PTIME on po-databases whose relations have either bounded width or bounded ia-width.*

To prove this, we show the following auxiliary result.

Lemma 52. *For any $\text{PosRA}^{\text{accGBy}}$ query $Q := \text{accumGroupBy}_{h,\oplus,P}(Q')$ and family $\mathcal{D}$ of po-databases, the CERT problem for Q on input po-databases from $\mathcal{D}$ reduces in PTIME to the CERT problem for $\text{accum}_{h,\oplus}(R)$ (where R is a relation name), on the family $\mathcal{D}'$ of po-databases mapping the name R to a subset of a po-relation of $\{Q'(D) \mid D \in \mathcal{D}\}$.*

Proof. To prove that, consider an instance of CERT for Q, defined by an input po-database D of $\mathcal{D}$ and candidate possible world L. We first evaluate $\Gamma' := Q'(D)$ in PTIME. Now, for each tuple value t in $\Pi_P(\Gamma')$, let Γ_t be the restriction of Γ' to the elements matching this value; note that the po-database mapping R to Γ_t is indeed in the family $\mathcal{D}'$. We solve CERT for $\text{accum}_{h,\oplus}(R)$ on each $R \mapsto \Gamma_t$ in

PTIME with the candidate possible world obtained from L by extracting the accumulation value for that group, and answer YES to the original CERT instance iff all these invocations answer YES. As this process is clearly in PTIME, it just remains to show correctness of the reduction.

For the forward direction, assume that each of the invocations answers YES, but the initial instance to CERT was negative. Consider two linear extensions of Γ' that achieve different accumulation results and witness that the initial instance was negative, and consider a group t where these accumulation results for these two linear extensions differ. Considering the restriction of these linear extensions to that group, we obtain the two different accumulation values for that group, so that the CERT invocation for Γ_t should not have answered YES.

For the backward direction, assume that the invocation for tuple t does not answer YES, then considering two witnessing linear extensions for that invocation, and extending them two linear extensions of Γ' by enumerating other tuples in an indifferent way, we obtain two different accumulation results for Q which differ in their result for t. This concludes the proof. □

This allows us to show Theorem 51.

Proof. We consider all tractability results of Section 6 in turn, and show that they extend to $\text{PosRA}^{\text{accGBy}}$ queries, under the same restrictions on operators, accumulation, and input po-relations.

First, we consider the tractability of CERT for accumulation in a cancellative monoid (Theorem 32). As this result holds for any input po-database, tractability for $\text{PosRA}^{\text{accGBy}}$ follows directly from Lemma 52.

Second, we consider the tractability of CERT for $\text{PosRA}^{\text{acc}}_{\text{LEX}}$ queries with a finite accumulation operator on po-databases of bounded width (Theorem 43). The result extends because, for any family $\mathcal{D}$ of po-databases whose po-relations have width at most k for some $k \in \mathbb{N}$, we know by Lemma 4 that the result $Q'(D)$ for $D \in \mathcal{D}$ also has width depending only on Q' and on k, and we know that restricting to a subset of $Q'(D)$ (namely, each group) does not increase the width (this is like the case of selection in the proof of Lemma 4). Hence, the family $\mathcal{D}'$ also has bounded width, and we can concludes using Lemma 52.

Third, we consider the tractability of CERT for $\text{PosRA}^{\text{acc}}_{\text{no}\times}$ queries with a finite and position-invariant accumulation operator on po-databases whose relations have either bounded width or bounded ia-width (Theorem 45). The result extends because, by Lemma 25 and subsequent observations, the result $Q'(D)$ for $D \in \mathcal{D}$ is a union of a po-relation of bounded width and of a po-relation with bounded ia-width. Restricting to a subset (i.e., a group), this property is preserved (as in the case of selection in the proof of Lemma 4 and of Lemma 26), which allows us to conclude using Lemma 52. □

7.2. 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 53. *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 54. *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 the totally ordered relation $\Gamma_2 := \Pi_{hotelname}(Hotel_2)$: the two "Mercure" tuples are an id-set. 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, then we call Γ *safe*, and set $\mathrm{dupElim}(\Gamma)$ 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 $\mathrm{dupElim}(\Gamma)$ 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 55. *In Example 54, we have* $\mathrm{dupElim}(\Gamma_1) = \emptyset$ *but* $\mathrm{dupElim}(\Gamma_2) = (\mathrm{Balzac}, \mathrm{Mercure})$.

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 56. *For any list relation L, we let Γ_L be a po-relation such that $pw(\Gamma_L) = \{L\}$. For Γ a po-relation, let* $\mathrm{dupElim}(\Gamma) := \bigcup_{L \in pw(\Gamma)} \mathrm{dupElim}(\Gamma_L)$. *We say that* $\mathrm{dupElim}(\Gamma)$ completely fails *if we have* $\mathrm{dupElim}(\Gamma) = \emptyset$*, i.e.,* $\mathrm{dupElim}(\Gamma_L) = \emptyset$ *for every $L \in pw(\Gamma)$.*

Example 57. *Consider the totally ordered po-relation Restaurant from Figure 1, and a totally ordered po-relation $Restaurant_2$ whose only possible world is* (Tsukizi, Gagnaire)*. Let $Q :=$* $\mathrm{dupElim}(\Pi_{restname}(Restaurant) \cup Restaurant_2)$*. 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).

We first define the notion of *quotient* of a po-relation by *value equality*.

Definition 58. *For a po-relation* $\Gamma = (ID, T, <)$*, we define the* value-equality quotient of Γ *as the directed graph* $\mathrm{G}_\Gamma = (ID', E)$*, where*

- ID' *is the quotient of* ID *by the equivalence relation* $id_1 \sim id_2 \Leftrightarrow T(id_1) = T(id_2)$*, i.e., it is a set of equivalence classes that are subsets of* ID*;*
- *The edge set* E *is defined by setting* $(id'_1, id'_2) \in E$ *for* $id'_1, id'_2 \in ID'$ *iff* $id'_1 \neq id'_2$ *and there are* $id_1 \in id'_1$ *and* $id_2 \in id'_2$ *such that* $id_1 < id_2$*.*

We claim that cycles in the value-equality quotient of Γ precisely characterize complete failure of dupElim.

Proposition 59. *For any po-relation* Γ*,* $\mathrm{dupElim}(\Gamma)$ *completely fails iff* G_Γ *has a cycle.*

Proof. Fix an input po-relation $\Gamma = (ID, T, <)$. We first show that the existence of a cycle implies complete failure of dupElim. Let $id'_1, \ldots, id'_n, id'_1$ be a simple cycle of G_Γ. For all $1 \leqslant i \leqslant n$, there exist $id_{1i}, id_{2i} \in id'_1$ such that $id_{2i} < id_{1(i+1)}$ (with the convention $id_{1(n+1)} = id_{11}$) and the $T(id_{2i})$ are pairwise distinct.

Let L be a possible world of Γ and let us show that dupElim fails on any po-relation Γ_L that *represents* L, i.e., $\Gamma_L = (ID_L, T_L, <_L)$ is totally ordered and $pw(\Gamma_L) = \{L\}$. Assume by contradiction that for all $1 \leqslant i \leqslant n$, id'_i forms an id-set of Γ_L. Let us show by induction on j that for all $1 \leqslant j \leqslant n$, $id_{21} \leqslant_L id_{2j}$, where $\leqslant_L$ denotes the non-strict order defined from $<_L$ in the expected fashion. The base case is trivial. Assume this holds for j and let us show it for $j+1$. Since $id_{2j} < id_{1(j+1)}$, we have $id_{21} \leqslant id_{2j} <_L id_{1(j+1)}$. Now, if $id_{2(j+1)} <_L id_{21}$, then $id_{2(j+1)} <_L id_{21} <_L id_{1(j+1)}$ with $T(id_{2(j+1)}) = T(id_{1(j+1)}) \neq T(id_{21})$, so this contradicts the fact that id'_{j+1} is an id-set. Hence, as L is a total order, we must have $id_{21} \leqslant_L id_{2(j+1)}$, which proves the induction case. Now the claim proved by induction implies that $id_{21} \leqslant_L id_{2n}$, and we had $id_{2n} < id_{11}$ in Γ and therefore $id_{2n} <_L id_{11}$, so this contradicts the fact that id'_1 is an id-set. Thus, dupElim fails in Γ_L. We have thus shown that dupElim fails in every possible world of Γ, so that it completely fails.

Conversely, let us assume that G_Γ is acyclic. Consider a topological sort of G_Γ as $id'_1, \ldots, id'_n$. For $1 \leqslant j \leqslant n$, let L_j be a linear extension of the poset $(id'_j, <_{|id'_j})$. Let L be the concatenation of $L_1, \ldots L_n$. We claim L is a linear extension of Γ such that dupElim does not fail in $\Gamma_L = (ID_L, T_L, <_L)$; this latter fact is clear by construction of L, so we must only show that L obeys the comparability relations of Γ. Now, let $id_1 < id_2$ in Γ. Either for some $1 \leqslant j \leqslant n$ we have $id_1, id_2 \in id'_j$, and then the tuple for id_1 precedes the one for id_2 in L_j by construction, so we have $t_1 <_L t_2$; or they are in different classes id'_{j_1} and id'_{j_2} and this is reflected in G_Γ, which means that $j_1 < j_2$ and $id_1 <_L id_2$. Hence, L is a linear extension, which concludes the proof. □

We can now state and prove the result.

Theorem 60. *For any po-relation* Γ*, we can test in PTIME if* $\mathrm{dupElim}(\Gamma)$ *completely fails; if it does not, then we can compute in PTIME a po-relation* Γ' *such that* $pw(\Gamma') = \mathrm{dupElim}(\Gamma)$*.*

Proof. We first observe that G_Γ can be constructed in PTIME, and that testing that G_Γ is acyclic is also done in PTIME. Thus, using Proposition 59, we can determine in PTIME whether dupElim(Γ) fails.

If dupElim(Γ) does not fail, then we let $\mathrm{G}_\Gamma = (ID', E)$ and construct the relation Γ' that will stand for dupElim(Γ) as $(ID', T', <')$, where $T'(id')$ is the unique $T'(id)$ for $id \in id'$ and $<'$ is the transitive closure of E, which is antisymmetric because G_Γ is acyclic. Observe that the underlying bag relation of Γ' has one identifier for each distinct tuple value in Γ, but has no duplicates.

Now, it is easy to check that $pw(\Gamma') = \text{dupElim}(\Gamma)$. Indeed, any possible world L of Γ' can be achieved in dupElim(Γ) by considering, as in the proof of Proposition 59, some possible world of Γ obtained following the topological sort of G_Γ defined by L. This implies that $pw(\Gamma') \subseteq \text{dupElim}(\Gamma)$.

Conversely, for any possible world L of Γ, dupElim(Γ_L) (for Γ_L a po-relation that represents L) fails unless, for each tuple value, the occurrences of that tuple value in Γ_L is an id-set. Now, in such an L, as the occurrences of each value are contiguous and the order relations reflected in G_Γ must be respected, L is defined by a topological sort of G_Γ (and some topological sort of each id-set within each set of duplicates), so that dupElim(Γ_L) can also be obtained as the corresponding linear extension of Γ'. Hence, we have $\text{dupElim}(\Gamma) \subseteq pw(\Gamma')$, proving their equality and concluding the proof. □

Last, we observe that dupElim can indeed be used to undo some of the effects of bag semantics.

Proposition 61. *For any po-relation* Γ*, we have* $\text{dupElim}(\Gamma \cup \Gamma) = \text{dupElim}(\Gamma)$*: in particular, one completely fails iff the other does.*

Proof. Let G_Γ be the value-equality quotient of Γ and G'_Γ be the value-equality quotient of $\Gamma \cup \Gamma$. It is easy to see that these two graphs are identical: any edge of G_Γ witnesses the existence of the same edge in G'_Γ, and conversely any edge in G'_Γ must correspond to a comparability relation between two tuples of one of the copies of Γ (and also in the other copy), so that it also witnesses the existence of the same edge in Γ. Hence, by Proposition 59, one duplicate elimination operation completely fails iff the other does. Further, by Theorem 60, we have indeed the equality that we claimed. □

We can also show that most of our previous tractability results Sections 4–6 still apply when the duplicate elimination operator is added. We first clarify the semantics of query evaluation when complete failure occurs: given a query Q in PosRA extended with dupElim, and given a po-database D, if complete failure occurs at any occurrence of the dupElim operator when evaluating $Q(D)$, then we set $pw(Q(D)) := \emptyset$, pursuant to our choice of defining query evaluation on po-relations as yielding all possible results on all possible worlds. If Q is a $\text{PosRA}^{\text{acc}}$ query extended with dupElim, we likewise say that its possible accumulation results are $\emptyset$.

This implies that for any PosRA query Q extended with dupElim, for any input po-database D, and for any candidate possible world v, the `POSS` and `CERT`

problems for Q are vacuously false on instance (D, v) if complete failure occurs at any stage when evaluating $Q(D)$. The same holds for PosRA$^{\text{acc}}$ queries.

Theorem 62. *Theorems 20, 32, 43 and Proposition 48 extend to PosRA and PosRA$^{\text{acc}}$ where we allow* dupElim *(but impose the same restrictions on query operators, accumulation, and input po-relations). Specifically:*

- *For any fixed $k \in \mathbb{N}$ and fixed PosRA$_{\texttt{LEX}}$ query Q which may additionally use* dupElim*, the* `POSS` *problem for Q in in PTIME on po-databases of bounded width.*
- *For any PosRA$^{\text{acc}}$ query Q which may additionally use* dupElim *and where accumulation is performed in a cancellative monoid, the* `CERT` *problem for Q is in PTIME.*
- *For any PosRA$^{\text{acc}}_{\texttt{LEX}}$ query Q which may additionally use* dupElim *and where the accumulation operator is finite, the* `POSS` *and* `CERT` *problems are in PTIME on po-databases of bounded width.*
- *For any PosRA query which may additionally use the* dupElim *operator, the problems* ***select-at-k****,* ***top-k****, and* ***tuple-level comparison*** *are in PTIME.*

To prove this result, observe that these four results are proved by first evaluating the query result in PTIME using Proposition 2. So we can still evaluate the query in PTIME, using in addition Theorem 60. Either complete failure occurs at some point in the evaluation, and we can immediately solve `POSS` and `CERT` by our initial remark above, or no complete failure occurs and we obtain in PTIME a po-relation Γ on which to solve `POSS` and `CERT`. Hence, in what follows, we can assume that no complete failure occurs at any stage.

It is then immediate that Theorem 32 and Proposition 48 still apply, because they did not make any assumptions on the po-relation Γ on which they applied. As for Theorems 20 and 43, the only assumption that they made on Γ is that its width was constant. Hence, we can conclude the proof of Theorem 62 from the following width preservation result.

Lemma 63. *For any constant $k \in \mathbb{N}$ and po-relation Γ of width $\leqslant k$, if* dupElim(Γ) *does not completely fail, then it has width $\leqslant k$.*

Proof. It suffices to show that to every antichain A of dupElim(Γ), there is an antichain A' of the same cardinality in Γ. Construct A' by picking a member of each of the classes of A. Assume by contradiction that A' is not an antichain, hence, there are two tuples $t_1 < t_2$ in A', and consider the corresponding classes id_1 and id_2 in A. By our characterization of the possible worlds of dupElim(Γ) in the proof of Theorem 60 as obtained from the topological sorts of the value-equality quotient G_Γ of Γ, as $t_1 < t_2$ implies that (id_1, id_2) is an edge of G_Γ, we conclude that we have $id_1 < id_2$ in A, contradicting the fact that it is an antichain. □

We have just shown in Theorem 62 that our tractability results still apply when we allow the duplicate elimination operator. 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 17).

Theorem 64. *For any PosRA query Q, `POSS` and `CERT` for* dupElim(Q) *are in PTIME.*

Proof. Let D be an input po-relation, and L be the candidate possible world (a list relation). We compute the po-relation Γ' such that $pw(\Gamma') = Q(D)$ in PTIME using Proposition 2 and the po-relation $\Gamma := \text{dupElim}(\Gamma')$ in PTIME using Theorem 60. If duplicate elimination fails, then we vacuously reject for `POSS` and `CERT`. Otherwise, by the definition of dupElim, the resulting po-relation Γ is such that each tuple value is realized exactly once. Note that we can reject immediately if L contains multiple occurrences of the same tuple, or does not have the same underlying set of tuples as Γ; so we assume that L has the same underlying set of tuples as Γ and no duplicate tuples.

The `CERT` problem is in PTIME on Γ by Theorem 32, so we need only study the case of `POSS`, namely, decide whether $L \in pw(\Gamma)$. Let Γ_L be a po-relation that represents L. As Γ_L and Γ have no duplicate tuples, there is only one way to match each identifier of Γ_L to an identifier of Γ. Build Γ'' from Γ by adding, for each pair $id_i <_L id_{i+1}$ of consecutive tuples of Γ_L, the order constraint $id''_i <'' id''_{i+1}$ on the corresponding identifiers in Γ''. We claim that $L \in pw(\Gamma)$ iff the resulting Γ'' is a po-relation, i.e., its transitive closure is still antisymmetric, which can be tested in PTIME by computing the strongly connected components of Γ'' and checking that they are all trivial.

To see why this works, observe that, if the result Γ'' is a po-relation, it is a total order, and so it describes a way to achieve L as a linear extension of Γ because it does not contradict any of the comparability relations of Γ. Conversely, if $L \in pw(\Gamma)$, assuming to the contrary the existence of a cycle in Γ'', we observe that such a cycle must consist of order relations of Γ and Γ_L, and the order relations of Γ are reflected in Γ_L as it is a linear extension of Γ, so we deduce the existence of a cycle in Γ_L, which is impossible by construction. Hence, we have reached a contradiction, and we deduce the desired result. □

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. A natural desideratum for our semantics on (partially) ordered relations is that it should be a faithful extension of the bag semantics for relational algebra. We first consider the BALG^1 language on bags [18] (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, and 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 $\mathrm{bag}(\Gamma)$ its underlying bag relation, and define likewise $\mathrm{bag}(D)$ for a po-database D as the database of the underlying bag relations. For the following comparison, we identify both $\times_{\texttt{DIR}}$ and $\times_{\texttt{LEX}}$ with the $\times$ of [18] (as both our product operations yield the same bag as output, for any input), and we identify our union with the additive union of [18]. The following then trivially holds.

Proposition 65. *For any PosRA query Q and a po-relation D,* $\mathrm{bag}(Q(D)) = Q(\mathrm{bag}(D))$*, where $Q(D)$ is defined according to our semantics and $Q(\mathrm{bag}(D))$ is defined by $BALG^1_+$.*

Proof. There is an exact correspondence in terms of the output bags between additive union and our union; between cross product and $\times_{\texttt{DIR}}$ and $\times_{\texttt{LEX}}$; between our selection and that of BALG^1_+, and similarly for projection (as noted before the statement of Proposition 65 in the main text, a technical subtlety is that the projection of BALG can only project on a single attribute, but one can encode "standard" projection on multiple attributes). The proposition follows by induction on the query structure. □

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 $\mathrm{PosRA}^{\mathrm{acc}}$ and BALG^1 are incomparable in terms of expressive power.

A better yardstick to compare against for accumulation could be the work of [19]: 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 $\mathcal{NRL}^{\mathrm{aggr}}$ on nested relations. On flat relations, $\mathcal{NRL}^{\mathrm{aggr}}$ 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 $\texttt{POSS}/\texttt{CERT}$ are trivial otherwise). On the other hand, $\mathcal{NRL}^{\mathrm{aggr}}$ cannot test parity (Corollary 5.7 in [19]) 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, 21], which also attempt to bridge partial order theory and data management (although, again, these works do not study possibility and certainty). *Pomsets* are labeled posets quotiented by isomorphism (i.e., renaming of identifiers), like po-relations. Beyond similarities in the language design, a major conceptual difference between our formalism and that of [20, 21] is that their work 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 65 (see, for instance, the semantics of duplicate elimination 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, then the resulting fragment $\mathcal{P}$om-$\mathcal{A}\text{lg}_{\varepsilon_n}$ of the language of [20] can yield only series-parallel outputs (Proposition 4.1 of [20]), unlike PosRA queries whose output order may be arbitrary. To formalize this, we need the notion of a *realizer* [5] of a poset $P = (V, <)$: this is a set of total orders $(V, <_1), \ldots, (V, <_n)$ such that, for every $x, y \in V$, we have $x < y$ iff $x <_i y$ for all i. We can use realizers to express arbitrary po-relations using the $\times_{\texttt{DIR}}$-product, as is shown by rephrasing in our context an existing result on partial orders (Theorem 9.6 of [22], see also [23]).

Lemma 66. *Let $n \in \mathbb{N}$, and let $(P, <_P)$ be a poset that has a realizer $(L_1, \ldots, L_n)$ of size n. Then P is isomorphic to a subset Γ' of $\Gamma = [\leqslant l] \times_{\texttt{DIR}} \cdots \times_{\texttt{DIR}} [\leqslant l]$, with n factors in the product, for some integer $l \in \mathbb{N}$ (the order on Γ' being the restriction on that of Γ).*

Proof. We define Γ by taking $l := |P|$, and we identify each element x of P to $f(x) := (n_1^x, \ldots, n_n^x)$, where n_i^x is the position where x occurs in L_i. Now, for any $x, y \in P$, we have $x <_P y$ iff $n_i^x < n_i^y$ for all $1 \leqslant i \leqslant n$ (that is, $x <_{L_i} y$), hence iff $f(x) <_\Gamma f(y)$: this is because there are no two elements $x \neq y$ and $1 \leqslant i \leqslant n$ such that the i-th components of $f(x)$ and of $f(y)$ are the same. Hence, taking Γ' to be the image of f (which is injective), Γ' is indeed isomorphic to P. □

This implies that PosRA queries can yield arbitrary po-relations as output.

Proposition 67. *For any po-relation Γ, there is a PosRA query Q with no inputs such that $Q() = \Gamma$.*

Proof. We first show that for any poset $(P, <)$, there exists a $\text{PosRA}_{\texttt{DIR}}$ query Q such that the tuples of $\Gamma' := Q()$ all have unique values and the underlying poset of Γ' is $(P, <)$. Indeed, we can take d to be the *order dimension* of P, which is necessarily finite [5], and then, by definition, P has a realizer of size d. By Lemma 66, there is an integer $l \in \mathbb{N}$ such that $\Gamma'' := [\leqslant l] \times_{\texttt{DIR}} \cdots \times_{\texttt{DIR}} [\leqslant l]$ (with n factors in the product) has a subset S isomorphic to $(P, <)$. Hence, letting ψ be a tuple predicate such that $\sigma_\psi(\Gamma'') = S$ (which can clearly be constructed by enumerating the elements of S), the query $Q' := \sigma_\psi(\Gamma'')$ proves the claim, with Γ'' expressed as above.

Now, to prove the desired result from this claim, build Q from Q' by taking its join (i.e., $\times_{\texttt{LEX}}$-product, selection, projection) with a union of singleton constant expressions that map each unique tuple value of $Q'()$ to the desired value of the corresponding tuple in the desired po-relation Γ. This concludes the proof. □

We conclude that $\mathcal{P}\text{om-}\mathcal{A}\text{lg}_{\varepsilon_n}$ does not subsume PosRA.

Incompleteness in databases. Our work is inspired by the field of incomplete information management, which has been studied for various models [24, 12], in particular relational databases [25]. This field inspires our design of po-relations and our study of possibility and certainty [13, 26]. However, uncertainty in these settings typically focuses on *whether* tuples exist or on what their *values* are (e.g., with nulls [27], including the novel approach of [28, 29]; with c-tables [25], probabilistic databases [30] or fuzzy numerical values as in [31]). 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 [32, 33, 29] 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 [34, 35, 36, 37, 38]. This is in particular the case in works aiming to query sequences or support iteration (see e.g. [37, 38], which however do not consider uncertainty and consequently neither partial order). However, our goal, setting and perspective are 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 [36, 39, 37]. Some works also study uncertainty on totally ordered *numerical* domains [31, 40], while we look at general order relations.

Temporal databases. *Temporal databases* [41, 42] consider order on facts, but it is usually induced by timestamps, hence total. A notable exception is [43] 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 [44, 45, 46, 47, 48], with some operators being the same as ours, and for *rank aggregation* [49, 44, 50], 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 [49], or a preference function [47]). In contrast, we focus on maintaining a faithful model of *all* possible worlds without reconciling them, studying possible and certain answers in this respect.

Computational social choice. The notion of preferences has been studied in the domain of computational social choice to determine the possible outcomes of an election given partial preference information expressed by voters [51]. In this setting, the notions of *possible winners* and *necessary winners* have been introduced to summarize the possible outcomes, and they have been connected to the notion of possible and necessary answers of database queries [52]. The complexity of these problems has been studied, with a dichotomy result that classifies its complexity depending on the aggregation used [51, 53, 54, 55]. However, the expressiveness of this computational social choice framework is incomparable to that of our framework. Specifically, their framework only studies possible and necessary answers in terms of achieving a maximal score computed as a sum of numerical values following some positional scoring rule, whereas our framework can perform accumulation in arbitrary monoids and on top of positive relational algebra queries. Conversely, there is no apparent way in our framework to encode accumulation following positional scoring rules, as we would need to apply accumulation to sum the candidate scores, and then look at the top answers according to a different order on the result.

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 several tractable cases.

A prime motivation for our work is to provide a semantics for a fragment of SQL (namely, SPJU+aggregates) in presence of partially ordered data. We see our work as a first step in this respect, and our choice of operators for the algebra is by no means the only possible one. In future work we plan to study the incorporation of additional operators, including in particular constructors of the (partial) order based on the tuple values. We will also investigate how to combine order-uncertainty with uncertainty on values, and study additional

semantics for dupElim (to avoid the pitfalls of the proposed semantics which we discussed above).

In connection with the choice of operators, a natural question is whether one may achieve a completeness result. We have shown (Proposition 67) that our language is complete in terms of "individual outputs", i.e., that PosRA can be used to construct any po-relation using only the built-in constant relations and operators. A more challenging goal is to design a language that is complete in terms of transformations, i.e. that may capture all functions over po-relations in some class. This is another intriguing topic for further investigation.

Last, many open questions remain about the complexity of `POSS`, e.g., we do not know whether `POSS` is tractable when the accumulation monoid is a finite group. Ideally, we would want to establish a dichotomy result for the complexity of `POSS`, and a complete syntactic characterization of cases where `POSS` is tractable: this is investigated further in a follow-up work involving the first author [56].

Acknowledgments. We are grateful to Marzio De Biasi, to Pálvölgyi Dömötör, and to Mikhail Rudoy, from `cstheory.stackexchange.com`, for helpful suggestions. We are also grateful to the anonymous reviewers for their feedback that helped improve this paper. This research was partially supported by the Israeli Science Foundation (grant 1636/13), the Blavatnik ICRC, and Intel.

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of databases, Addison-Wesley, 1995.

[2] A. Amarilli, M. L. Ba, D. Deutch, P. Senellart, Possible and certain answers for queries over order-incomplete data, in: Proc. TIME, 2017.

[3] L. S. Colby, E. L. Robertson, L. V. Saxton, D. V. Gucht, A query language for list-based complex objects, in: PODS, 1994.

[4] L. S. Colby, L. V. Saxton, D. V. Gucht, Concepts for modeling and querying list-structured data, Information Processing & Management 30 (1994).

[5] B. Schröder, Ordered Sets: An Introduction, Birkhäuser, 2003.

[6] D. R. Fulkerson, Note on Dilworth's decomposition theorem for partially ordered sets, in: Proc. Amer. Math. Soc, 1955.

[7] R. P. Dilworth, A decomposition theorem for partially ordered sets, Annals of Mathematics (1950).

[8] A. Brandstädt, V. B. Le, J. P. Spinrad, Posets, in: Graph Classes. A Survey, SIAM, 1987.

[9] R. P. Stanley, Enumerative Combinatorics, Cambridge University Press, 1986.

[10] J. M. Howie, Fundamentals of semigroup theory, Oxford: Clarendon Press, 1995.

[11] M. Lenzerini, Data integration: A theoretical perspective, in: PODS, 2002.

[12] L. Libkin, Data exchange and incomplete information, in: PODS, 2006.

[13] L. Antova, C. Koch, D. Olteanu, World-set decompositions: Expressiveness and efficient algorithms, in: ICDT, 2007.

[14] M. K. Warmuth, D. Haussler, On the complexity of iterated shuffle, J. Comput. Syst. Sci. 28 (1984).

[15] M. R. Garey, D. S. Johnson, Computers And Intractability. A Guide to the Theory of NP-completeness, W. H. Freeman, 1979.

[16] R. Fraîssé, L'intervalle en théorie des relations; ses généralisations, filtre intervallaire et clôture d'une relation, North-Holland Math. Stud. 99 (1984).

[17] J.-E. Pin, Syntactic semigroups, in: Handbook of Formal Languages, Springer, 1997.

[18] S. Grumbach, T. Milo, Towards tractable algebras for bags, J. Comput. Syst. Sci. 52 (1996).

[19] L. Libkin, L. Wong, Query languages for bags and aggregate functions, J. Comput. Syst. Sci. 55 (1997).

[20] S. Grumbach, T. Milo, An algebra for pomsets, in: ICDT, 1995.

[21] S. Grumbach, T. Milo, An algebra for pomsets, Inf. Comput. 150 (1999).

[22] T. Hiraguchi, On the dimension of orders, Sci. rep. Kanazawa Univ. 4 (1955).

[23] O. Øre, Partial order, in: Theory of Graphs, AMS, 1962.

[24] P. Barceló, L. Libkin, A. Poggi, C. Sirangelo, XML with incomplete information, J. ACM 58 (2010).

[25] T. Imieliński, W. Lipski, Incomplete information in relational databases, J. ACM 31 (1984).

[26] W. Lipski, Jr., On semantic issues connected with incomplete information databases, TODS 4 (1979).

[27] E. F. Codd, Extending the database relational model to capture more meaning, TODS 4 (1979).

[28] L. Libkin, Incomplete data: What went wrong, and how to fix it, in: PODS, 2014.

[29] L. Libkin, SQL's three-valued logic and certain answers, in: ICDT, 2015.

[30] D. Suciu, D. Olteanu, C. Ré, C. Koch, Probabilistic Databases, Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2011.

[31] M. A. Soliman, I. F. Ilyas, Ranking with uncertain scores, in: ICDE, 2009.

[32] P. Buneman, A. Jung, A. Ohori, Using powerdomains to generalize relational databases, TCS 91 (1991).

[33] L. Libkin, A semantics-based approach to design of query languages for partial information, in: Semantics in Databases, 1998.

[34] N. Immerman, Relational queries computable in polynomial time, Inf. Control 68 (1986).

[35] W. Ng, An extension of the relational data model to incorporate ordered domains, TODS 26 (2001).

[36] R. van der Meyden, The complexity of querying indefinite data about linearly ordered domains, J. Comput. Syst. Sci. 54 (1997).

[37] A. J. Bonner, G. Mecca, Sequences, datalog, and transducers, J. Comput. Syst. Sci. 57 (1998) 234–259.

[38] N. Coburn, G. E. Weddell, A logic for rule-based query optimization in graph-based data models, in: DOOD, 1993, pp. 120–145.

[39] M. Benedikt, L. Segoufin, Towards a characterization of order-invariant queries over tame graphs, Journal of Symbolic Logic 74 (2009).

[40] M. A. Soliman, I. F. Ilyas, S. Ben-David, Supporting ranking queries on uncertain and incomplete data, VLDBJ 19 (2010).

[41] J. Chomicki, D. Toman, Time in database systems, in: Handbook of Temporal Reasoning in Artificial Intelligence, Elsevier, 2005.

[42] R. T. Snodgrass, J. Gray, J. Melton, Developing time-oriented database applications in SQL, Morgan Kaufmann, 2000.

[43] W. Fan, F. Geerts, J. Wijsen, Determining the currency of data, TODS 37 (2012).

[44] M. Jacob, B. Kimelfeld, J. Stoyanovich, A system for management and analysis of preference data, VLDB Endow. 7 (2014).

[45] A. Arvanitis, G. Koutrika, PrefDB: Supporting preferences as first-class citizens in relational databases, IEEE TKDE 26 (2014).

[46] W. Kiessling, Foundations of preferences in database systems, in: VLDB, 2002.

[47] B. Alexe, M. Roth, W.-C. Tan, Preference-aware integration of temporal data, PVLDB 8 (2014).

[48] K. Stefanidis, G. Koutrika, E. Pitoura, A survey on representation, composition and application of preferences in database systems, TODS 36 (2011).

[49] R. Fagin, A. Lotem, M. Naor, Optimal aggregation algorithms for middleware, in: PODS, 2001.

[50] C. Dwork, R. Kumar, M. Naor, D. Sivakumar, Rank aggregation methods for the Web, in: WWW, 2001.

[51] K. Konczak, J. Lang, Voting procedures with incomplete preferences, in: IJCAI-05 Workshop on Advances in Preference Handling, 2005.

[52] B. Kimelfeld, P. G. Kolaitis, J. Stoyanovich, Computational social choice meets databases, in: Proc. IJCAI, 2018.

[53] L. Xia, V. Conitzer, Determining possible and necessary winners given partial orders, JAIR 41 (2011).

[54] N. Betzler, B. Dorn, Towards a dichotomy for the possible winner problem in elections based on scoring rules, J. Comput. Syst. Sci. 76 (2010).

[55] D. Baumeister, J. Rothe, Taking the final step to a full dichotomy of the possible winner problem in pure scoring rules, Information Processing Letters 112 (2012).

[56] A. Amarilli, C. Paperman, Topological sorting under regular constraints, in: ICALP, 2018.