Possible and Certain Answers for Queries over Order-Incomplete Data

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

1 **LTCI, CNRS, Télécom ParisTech, Université Paris-Saclay; Paris, France**
`first.last@telecom-paristech.fr`
2 **Qatar Computing Research Institute, HBKU; Doha, Qatar**
`mlba@qf.org.qa`
3 **Blavatnik School of Computer Science, Tel Aviv University; Tel Aviv, Israel**
`danielde@post.tau.ac.il`
4 **IPAL, CNRS, National University of Singapore; Singapore**

Abstract

To combine and query ordered data originating from multiple sources, one needs a framework that can handle uncertainty about the possible orderings. Examples of such "order-incomplete" data include lists of properties (such as hotels and restaurants) ranked by an unknown function reflecting relevance or customer ratings; documents edited concurrently with uncertainty on the order of contributions; and the result of integrating event sequences such as log entries. This paper introduces a query language for order-incomplete data, based on the positive relational algebra, augmented with an accumulation operator to perform order-aware aggregation. We use partial orders as a representation system, and study possible and certain answers for queries in this context. In their general form, possibility and certainty are shown to be NP-complete and coNP-complete, respectively. However, we identify a large class of cases for which the problems are tractable, based on fine-grained characterizations of the partial orders that query evaluation may produce. Last, we introduce an operator that merges identical tuples (possibly appearing with different orderings), in the spirit of set semantics, and revisit our results.

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

This paper studies query evaluation over order-incomplete data in a relational setting. We focus on the running example of restaurants and hotels from travel websites, ranked according to proprietary functions. An example query could compute the union of lists of restaurants, each from a distinct website, and further ask for the ordered list of restaurant–hotel pairs such that the restaurant and hotel are in the same district. As we do not know how the proprietary order is defined, the result of transformations may become *uncertain*: in our example, there may be multiple reasonable orderings of restaurants in the union result, or multiple orderings of restaurant–hotel pairs. Further, we may apply an order-aware *accumulation function* to the result, e.g., extracting only the highest ranked such pairs, concatenating (a subset of) their names, or assessing the attractiveness of a particular district as a function of its high-ranked restaurants. Each possible order may yield a different accumulation result.

Main contributions. We introduce a query language with accumulation for order-incomplete data, and then undertake what is, to our knowledge, the first general study of the *complexity of possible and certain answers for queries over such data.* We show that these problems are intractable in general, but identify multiple realistic tractable classes. Importantly, we do not assume that a decisive choice of order can be made, unlike, e.g., rank aggregation [17]. Instead, we evaluate queries by representing *all possible results*, i.e., all those that are consistent with the individual input orders.

Our order-incomplete relations are essentially equivalent to *labeled posets*, or *pomsets* [22], and our complexity results on possibility and certainty imply similar results on testing whether a label sequence is achieved as a linear extension of a labeled poset (also including accumulation in monoids). We study this problem under bounds on order-theoretic parameters of the input (e.g., poset width [38] or a new measure of *ia-width*), and examine how the bounds are preserved by our query language. To our knowledge, such complexity results on labeled posets were not known before, and they may be of independent interest. These results do not follow from existing results on *posets*, because of label ambiguity, as illustrated in Example 12. We explain in more detail in the related work section (Section 9) how our results relate to labeled posets (in particular to [22]), but we will present them using relational algebra terminology, to match our intended application to ordered data integration.

We next overview the main parts of our study. Full proofs are provided in the appendix.

Model (Sections 2-4). Our data model relies on *bag* relations, and we equip each relation with a *partial order* over its tuples: we call this a *po-relation*. Our use of bags means, in our example, that we keep every occurrence of each hotel, because they may appear at different order positions; duplicate consolidation where possible, is discussed in Section 8. Using notions from order theory, we then define a semantics for the positive relational algebra (PosRA), adapted to po-relations: selection and projection do not affect order, while union is the *parallel composition* [8] of posets, i.e., keeps only the order constraints among tuples from the same input relation. For product, we introduce two operators: *direct product* [42] (two tuples in the product are comparable iff both components compare in the same way in the input relations); and *lexicographic product* (follow the order in the first component and use the second to break ties). The resulting language can capture other operators, e.g., *series composition* (concatenation). Each linear extension of a po-relation leads to a totally ordered *possible world*, and we show that po-relations form a *strong representation system* for PosRA: the uncertain result of a query on a po-database can always be represented as a po-relation.

We extend PosRA to PosRA$^{\text{acc}}$, which allows order-aware *accumulation* (generalizing aggregation) as the last operation. On totally ordered relations, accumulation maps the tuples to a monoid and aggregates them with the associative monoid operator The possible accumulation results on a po-relation are those that can be obtained on its possible worlds.

We then introduce the problems of possible (`POSS`) and certain (`CERT`) answers with respect to query results. We show that different choices of accumulation functions can capture different notions of interest, such as the possibility and certainty of a tuple appearing in a particular location or before another tuple.

Complexity Analysis (Sections 5–7). Our main technical contribution is the complexity analysis of the `POSS` and `CERT` problems for PosRA and PosRA$^{\text{acc}}$. As possibility and certainty of a tuple position are in PTIME, we study possibility and certainty of outputs (for PosRA) and accumulation results (for PosRA$^{\text{acc}}$). For PosRA, `POSS` is NP-complete but `CERT` is PTIME. For PosRA$^{\text{acc}}$, `CERT` becomes coNP-complete.

These hardness results lead us to study realistic problem restrictions where `POSS` and `CERT` can be solved efficiently, without enumerating the (possibly exponential) number of possible worlds. We start with restrictions for PosRA that ensure the tractability of `POSS`. These are achieved by bounding the "level of uncertainty" in the *input*, and the operators allowed. Specifically, if all input relations are totally ordered and the direct product is disallowed, then `POSS` is in PTIME (but hardness holds if we do not disallow the direct product): this covers the application case where the order on the sources is completely known. Similarly, querying unordered relations (and imposing order only via the query) is tractable for all PosRA, covering the case where order is completely unknown. These results generalize to cases where the *width* of the input partial orders is bounded (i.e., "almost total" orders), and likewise for the *ia-width*, a novel measure on posets that covers "almost empty" orders.

We then study tractable restrictions for PosRA$^{\text{acc}}$, for both `POSS` and `CERT`, assuming a PTIME accumulation operator. We first show `CERT` (but not `POSS`) is in PTIME for *cancellative* monoids, which generalize groups and cover many accumulation operators. Extending our "uncertainty level" restrictions to PosRA$^{\text{acc}}$, we further prove that `POSS` and `CERT` are PTIME when width or ia-width is bounded (under technical conditions on the accumulation operator).

Duplicate Consolidation (Section 8). We conclude by studying the *consolidation* of duplicate tuples, with a dupElim operator. As duplicate tuples may have irreconcilable order relations with respect to other tuples, we allow dupElim to *fail* on some inputs (we also consider alternative semantics that avoid failure, and illustrate their pitfalls). We show that failure on po-relations can be detected in PTIME, that po-relations are still a strong representation system when there is no failure, and that all complexity results go through.

2 Data Model and PosRA

We revisit basic notions from databases and order theory and use them to define our model.

Relations. We fix a countable set of values $\mathcal{D}$ that includes $\mathbb{N}$ and infinitely many values not in $\mathbb{N}$. A *tuple* t *over* $\mathcal{D}$ of *arity* $\mathrm{a}(t)$ is an element of $\mathcal{D}^{\mathrm{a}(t)}$, denoted $\langle v_1, \ldots, v_{\mathrm{a}(t)} \rangle$. The *concatenation* of two tuples t_1 and t_2 is denoted $\langle t_1, t_2 \rangle$. We consider relations that are *bags* of tuples with unique identifiers and the same arity (referred to as the relation arity). Thus, a relation R is formally a pair (ID, T) where ID is a set of identifiers and T is a mapping from ID to tuples of the relation arity. The mapping need not be injective, so multiple copies of a tuple may appear in the relation, with different identifiers.

Isomorphisms of relations. While we use unique tuple identifiers to distinguish copies of the same tuple value (following our bag semantics), we do *not* assume that identifiers appear as an attribute that can be accessed by queries. Consequently, we *always* consider relations up to isomorphism of identifiers, where two relations $R = (ID, T)$ and $R' = (ID', T')$ are *isomorphic* if there is a bijection $\varphi : ID \mapsto ID'$ such that $T(id) = T'(\varphi(id))$ for all $id \in ID$.

We fix a *schema* $\mathcal{S}$, i.e., a set of relation names and arities, with an attribute name for each position of each relation. A *database* D is a set of relations over $\mathcal{S}$ and $\mathcal{D}$, every pair of relations having disjoint sets of identifiers (as we can always ensure by renaming identifiers).

List relations. A first step to introduce order on tuples is to consider *list relations* [12, 13], i.e., impose a *total order* over the identifiers of tuples in the relation: as we work with bags,

restname	*distr*
Gagnaire	8 ↓
TourArgent	5

(a) *Rest* table

restname	*distr*
Tsukizi	6

(b) $Rest_2$ table

hotelname	*distr*
Mercure	5 ↓
Balzac	8
Mercure	12

(c) *Hotel* table

Figure 1 Running example: Paris restaurants and hotels

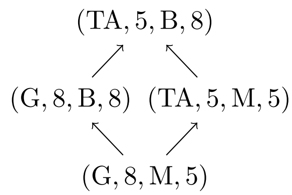

Figure 2 Example 3

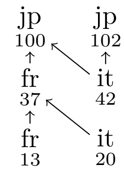

Figure 3 Example 12

the order is on *identifiers*, and multiple copies of a tuple may appear in different positions. However, when unioning or joining list relations, output tuples can be ordered in many ways, so that the result can no longer be represented as a list relation:

▶ **Example 1.** The database in Figure 1 contains information about restaurants and hotels in Paris. Tuples in each relation are totally ordered (top to bottom, following the arrows) by customer ratings from a given travel website, each relation coming from a different site.

When attempting to *union* *Rest* and $Rest_2$, we know nothing about the relative order between, e.g., ⟨Tsukizi, 6⟩ and ⟨Gagnaire, 8⟩. Similarly, if we *join* *Rest* and *Hotel*, there are multiple plausible ways to decide a relative order between pairs of restaurants and hotels.

There are two ways to handle this. The first is to enforce a *single* choice of order for the output, for instance interpreting union as concatenation and product as lexicographic order over the joined tuples [35], or making a preference-aware choice [2]. We follow a second approach: we represent *all* possible orderings through a partial order [15], as we now discuss.

Po-relations. We represent relations equipped with a partial order as *po-relations*:

▶ **Definition 2.** A *partially ordered relation*, or *po-relation* for short, is a triple $\Gamma = (ID, T, <)$, where $R = (ID, T)$ is the *underlying relation* of Γ and $<$ is a partial order over ID. The *possible worlds* of Γ are the list relations $pw(\Gamma) = \{(R, <_1), (R, <_2), \ldots, (R, <_n)\}$ where $<_1, ..., <_n$ are the linear extensions[1] of $<$. Note that, as Γ may contain multiple tuples with the same values, it may be the case that two different linear extensions $<_i$ and $<_j$ (which are defined on *identifiers*) are such that $(R, <_i)$ and $(R, <_j)$ are *isomorphic* list relations.

If $<$ is empty (i.e., imposes no order constraints), we call Γ *unordered*. If $<$ is total, we call Γ *totally ordered* and we can see it as a list relation $(t_1, ..., t_n)$. A *po-database* D is a set of po-relations with distinct relation names and disjoint identifiers: its possible worlds $pw(D)$ are obtained by choosing a possible world (i.e., a list relation) for each po-relation in D.

Po-relations are thus a way to model uncertainty over the order of tuples. They can equivalently be thought of as *labeled partial orders* or *pomsets* [36, 22], where the labels are tuples. Note that there is no uncertainty on the *value* of tuples in po-relations, but only on their *order*: the underlying relation is always certain.

Query language. We now introduce our query language for po-relations. We start with PosRA, i.e., the *positive relational algebra*, adapted to the partial-order setting. We also support an notion of *accumulation (as a last operation)*, which we present in the next section.

In our setting, the *selection* operator restricts the relation to a subset of its tuples, and the order on them is the restriction of the input order relation. The *tuple predicates* are (in)equalities over tuple attributes and/or values in $\mathcal{D}$, and Boolean combinations thereof.

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

[1] A *linear extension* $<_i$ of $<$ is a total order on the domain of $<$ such that for all $x < y$ we have $x <_i y$ [8].

The *projection* operator changes the tuple values, but keeps the original tuple ordering in the result. Following our *bag semantics*, we do not remove duplicate tuples when projecting.

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

As for *union*, we impose the minimal order constraints that are compatible with those of the inputs. We use the *parallel composition* [8] of two partial orders $<$ and $<'$ on disjoint sets ID and ID', i.e., the partial order $<'' := (< \parallel <')$ on $ID \cup ID'$ defined by: every $id \in ID$ is incomparable for $<''$ with every $id' \in ID'$; for each $id_1, id_2 \in ID$, we have $id_1 <'' id_2$ iff $id_1 < id_2$; for each $id'_1, id'_2 \in ID'$, we have $id'_1 <'' id'_2$ iff $id'_1 <' id'_2$. We use this to define:

union: Let $\Gamma = (ID, T, <)$ and $\Gamma' = (ID', T', <')$ be two po-relations of the same arity, where ID and ID' are disjoint (as can be ensured by renaming). We define $\Gamma \cup \Gamma' := (ID \cup ID', T \cup T', < \parallel <')$, where $T \cup T'$ maps $id \in ID$ to $T(id)$ and $id' \in ID'$ to $T'(id')$.

Note that, when Γ and Γ' are totally ordered, in general $\Gamma \cup \Gamma'$ is not. One could alternatively impose a particular total order on $\Gamma \cup \Gamma'$, e.g., decide that all tuples of Γ precede those of Γ', leading to an interpretation of union as *series composition* or *concatenation*. As we show, this specific interpretation can be expressed in our query language instead.

We next introduce two possible *product operators*. First, the *direct product* [42] $<_{\texttt{DIR}} := (< \times_{\texttt{DIR}} <')$ of two partial orders $<$ and $<'$ on disjoint sets ID and ID' is defined by $(id_1, id'_1) <_{\texttt{DIR}} (id_2, id'_2)$ for each $(id_1, id'_1), (id_2, id'_2) \in ID \times ID'$ iff $id_1 < id_2$ and $id'_1 <' id'_2$. We define the *direct product* operator over po-relations accordingly: two tuples in the product are comparable only if *both components* of both tuples compare in the same way.

direct product: For any po-relations $\Gamma = (ID, T, <)$ and $\Gamma' = (ID', T', <')$ with disjoint ID and ID', we define $\Gamma \times_{\texttt{DIR}} \Gamma' := (ID \times ID', T \times T', < \times_{\texttt{DIR}} <')$, where $T \times T'$ maps each $(id, id') \in ID \times ID'$ to $(T(id), T'(id'))$.

Again, the direct product result may not be totally ordered even when the inputs are.

The second product operator uses the *lexicographic product* (or *ordinal product* [42]) of two partial orders $<$ and $<'$ on disjoint ID and ID', denoted $<_{\texttt{LEX}} := (< \times_{\texttt{LEX}} <')$, and defined by $(id_1, id'_1) <_{\texttt{LEX}} (id_2, id'_2)$ for all $(id_1, id'_1), (id_2, id'_2) \in ID \times ID'$ iff either $id_1 < id_2$, or $id_1 = id_2$ and $id'_1 <' id'_2$. This time, the result is totally ordered if the input relations are.

lexicographic product: $\Gamma \times_{\texttt{LEX}} \Gamma'$ is the po-relation $(ID \times ID', T \times T', < \times_{\texttt{LEX}} <')$.

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

const: for any tuple t, the singleton po-relation $[t]$ has only one tuple with value t; for any $n \in \mathbb{N}$, the po-relation $\mathbb{N}^*_{\leqslant n}$ is the totally ordered relation $(1, ..., n)$, with arity 1

▶ **Example 3.** Let $Q := \mathit{Rest} \times_{\texttt{DIR}} (\sigma_{distr \neq \text{“12”}}(\mathit{Hotel}))$. Q admits two possible worlds: $(\langle \mathrm{G}, 8, \mathrm{M}, 5\rangle, \langle \mathrm{G}, 8, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, 5, \mathrm{M}, 5\rangle, \langle \mathrm{TA}, 5, \mathrm{B}, 8\rangle)$, $(\langle \mathrm{G}, 8, \mathrm{M}, 5\rangle, \langle \mathrm{TA}, 5, \mathrm{M}, 5\rangle, \langle \mathrm{G}, 8, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, 5, \mathrm{B}, 8\rangle)$. In a sense, this is the minimal order on hotel–restaurant pairs that is *consistent* with the order on the individual lists: we do not know how to order two pairs, except when both their hotels and their restaurants compare in the same way. The resulting po-relation is represented by the Hasse diagram in Figure 2, ordered from bottom to top.

Consider now $Q' := \Pi(\sigma_{Rest.distr = Hotel.distr}(Q))$, where the projection Π projects out $\mathit{Hotel.distr}$. Its possible worlds 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 a query Q'' similarly to Q' but replacing $\times_{\texttt{DIR}}$ by $\times_{\texttt{LEX}}$ in Q, we obtain only one possible order, given by *Rest* (the leftmost product operand): $(\langle \mathrm{G}, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, \mathrm{M}, 5\rangle)$.

We can then show:

▶ **Theorem 4.** *No* PosRA *operator can be expressed through a combination of the others.*

In particular, the proof (in Appendix) shows that the two product operators are incomparable. To this end, we show that if we disallow $\times_{\mathtt{DIR}}$ then we get an output of a restricted form (i.e., it is *series-parallel*, if the input po-database also is). Conversely, we show that the full language can capture *concatenation* (justifying its absence from our language); however, if we disallow $\times_{\mathtt{LEX}}$, we can no longer capture concatenation.

Furthermore, the semantics admits a natural possible-worlds interpretation, which will be useful in the sequel. Let us accordingly define the *possible worlds* of a query:

▶ **Definition 5.** Let Q be a PosRA query and D be a po-database whose possible worlds (databases of list relations) are $pw(D) = \{D_1, ..., D_n\}$. We define $Q(D) := \{Q(D_1), ..., Q(D_n)\}$.

The following simple result indicates the soundness of our construction. In the terminology of incomplete databases, po-relations form a *strong representation system* for PosRA queries:

▶ **Proposition 6.** *For any* PosRA *query Q and po-database D, we can compute in polynomial time in D (the exponent depending on Q) a po-relation Γ such that $pw(\Gamma) = Q(D)$.*

3 Accumulation

We now enrich PosRA with order-aware *accumulation* as the last operation, inspired by *right accumulation* and *iteration* in list programming and databases, and *aggregation* in relational databases. We recall the notion of a *monoid*, to be used as the domain of aggregation (which may differ from the domain $\mathcal{D}$ of tuple values):

▶ **Definition 7.** A *monoid* $(\mathcal{M}, \oplus, \varepsilon)$, which we abbreviate as $\oplus$, is a set $\mathcal{M}$ with a neutral element $\varepsilon \in \mathcal{M}$ and a binary composition law $\oplus : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ such that:
- $\oplus$ is associative: for all $u, v, w \in \mathcal{M}$, we have: $(u \oplus v) \oplus w = u \oplus (v \oplus w)$;
- ε is neutral: for all $v \in \mathcal{M}$, $\varepsilon \oplus v = v \oplus \varepsilon = v$.

Some applications may simply use $\mathcal{M} = \mathcal{D}$ (i.e. the domain of tuple values) with some associative operation and neutral value; but we will also show cases below where $\mathcal{M} \neq \mathcal{D}$.

▶ **Definition 8.** Let $(\mathcal{M}, \oplus, \varepsilon)$ be a monoid and let $h : \mathcal{D} \times \mathbb{N}^* \to (\mathcal{M}, \oplus, \varepsilon)$ be a function which we call the *accumulation map*. We call $\mathrm{accum}_{h,\oplus}$ an *accumulation operator*, and define its result on a totally ordered relation $L = (t_1, \ldots, t_n)$ as: $\mathrm{accum}_{h,\oplus}(L) := h(t_1, 1) \oplus \cdots \oplus h(t_n, n)$. In particular, if L is empty then $\mathrm{accum}_{h,\oplus}(L) := \varepsilon$.

The accumulation operator thus uses the accumulation map h to map the tuples to the accumulation monoid $\mathcal{M}$, where accumulation is performed by repeated application of $\oplus$. In a sense, this captures the map-accumulation structure in LISP. Note that we allow the map h to also take into account the absolute rank of tuples in the ordered relation.

It is then easy to extend the semantics of accumulation to po-relations: the possible results are the results of applying accumulation to the individual possible worlds.

▶ **Definition 9.** For an accumulation operator $\mathrm{accum}_{h,\oplus}$ and po-relation Γ, we define: $\mathrm{accum}_{h,\oplus}(\Gamma) := \mathrm{accum}_{h,\oplus}(pw(\Gamma)) := \{\mathrm{accum}_{h,\oplus}(L) \mid L \in pw(\Gamma)\}$.

Complexity assumption. Our definition allows arbitrary accumulation monoids, but for practical purposes we must limit the complexity of accumulation. Throughout the paper we thus impose a restriction on the accumulation operator, which we call *PTIME-evaluability*: given any *totally ordered* relation L, we assume that we can compute $\mathrm{accum}_{h,\oplus}(L)$ in PTIME. This assumption ensures that accumulation in each individual possible world is tractable, so that accumulation does not cause hardness on its own. PTIME-evaluability is satisfied by all examples of accumulation functions in this paper.

The PosRA$^{\text{acc}}$ language. We now define the language PosRA$^{\text{acc}}$: it 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 are $Q(D) := \text{accum}_{h,\oplus}(Q'(D))$.

Accumulation captures "standard" order-oblivious aggregation functions, such as sum, max, min, etc., with the identity accumulation map and with the corresponding *commutative* monoid: in this case, the result of accumulation is always certain (i.e., there is only one possible result). In contrast, many useful functions depend on the order of tuples:

▶ **Example 10.** As a first example, let Ratings(*user*, *restaurant*, *rating*) be an *unordered* relation describing ratings given by users to restaurants, where each user rated each restaurant at most once. Consider a po-relation Relevance(user) giving a partially-known ordering of users to indicate the relevance of their reviews. We wish to take reviews into account depending on a PTIME-computable weight function w, where $w(i)$ assigns a nonnegative weight to the opinion of the i-th most relevant user. Consider the query $Q_1 := \text{accum}_{h_1,+}(\sigma(\text{Relevance} \times_{\texttt{LEX}} \text{Ratings}))$ where we define $h_1(t, n) := t.\mathit{rating} \times w(n)$, and where σ selects tuples that satisfy: $\mathit{restaurant} = \text{“Gagnaire”} \land \text{Ratings}.\mathit{user} = \text{Relevance}.\mathit{user}$. Q_1 gives the total rating of "Gagnaire", and each possible world of Relevance may lead to a different accumulation result.

As a second example, consider an unordered 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: $Q_2 := \text{accum}_{h_2,\text{concat}}(\Pi_{\mathit{hotel}}(Q_2'))$, where $Q_2' := \sigma_{\text{City}.\mathit{city}=\text{HotelCity}.\mathit{city}}(\text{City} \times_{\texttt{LEX}} \text{HotelCity})$, where $h_2(t, n) := t$, and where "concat" denotes standard string concatenation. Q_2 concatenates the hotel names according to the preference order on the city where they are located, allowing any possible order between hotels of the same city and between hotels in incomparable cities.

Finally, accumulation allows us to perform various kinds of *position-based selection*. Consider for instance the *top-k* operator, which retrieves a list of the first k tuples: for a po-relation, the set of possible results is all possible such lists. We can implement *top-k* as $\text{accum}_{h_3,\text{concat}}$ with $h_3(t, n)$ being (t) for $n \leqslant k$ and ε otherwise, and with "concat" being list concatenation. We can similarly compute *select-at-k*, i.e., return the tuple at position k, using $\text{accum}_{h_4,\text{concat}}$, with $h_4(t, n)$ being (t) for $n = k$ and ε otherwise. Defining $h_5(t, n) := (t)$, we can also define $\text{accum}_{h_5,\text{concat}}$, which is the *identity* accumulation operator over relations.

4 Possibility and Certainty

Evaluating a PosRA query Q on a po-database D yields a set of possible worlds (totally ordered relations), which we can represent as a po-relation by Proposition 6. For PosRA$^{\text{acc}}$ queries, which may perform arbitrary PTIME accumulation, we have no such representation, but we still have a set of possible query results.

In both cases, however, a natural question is whether a given result is *possible* or not, i.e., whether it is one of the possible query outputs. Likewise, we can ask whether a result is *certain*, namely, only this single result is possible. We formalize these problems as follows:

▶ **Definition 11 (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 is isomorphic to some $L' \in Q(D)$, i.e., whether L is a possible result of the query. The *certainty problem* (`CERT`) asks if $Q(D) = \{L'\}$ where L' is isomorphic to L, i.e., whether L is the only possible result.

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

Note a subtlety in the above definitions: the identifiers of the candidate result L have no reason to match the identifiers in $Q(D)$, which is why our problem is defined up to isomorphism of identifiers. What matters is tuple *values*, but, as they can occur multiple times in L and $Q(D)$, it is not easy to match them, as the following example illustrates:

▶ **Example 12.** Consider a po-relation $\Gamma = (ID, T, <)$ with $ID = \{id_{13}, id_{20}, id_{37}, id_{42}, id_{100}, id_{102}\}$, with $T(id_{13}) := (\text{Gagnaire}, \text{fr})$, $T(id_{20}) := (\text{Italia}, \text{it})$, $T(id_{37}) := (\text{TourArgent}, \text{fr})$, $T(id_{42}) := (\text{Verdi}, \text{it})$, $T(id_{100}) := (\text{Tsukizi}, \text{jp})$, $T(id_{102}) := (\text{Sola}, \text{jp})$, and with $id_{13} < id_{37}$, $id_{20} < id_{37}$, $id_{37} < id_{100}$, $id_{42} < id_{100}$, and $id_{42} < id_{102}$. Intuitively, Γ describes a preference relation over restaurants, indicating their name and the nationality of their cuisine. Consider $Q := \Pi(\Gamma)$ that projects Γ on nationality; we illustrate the result (with the original identifiers) in Figure 3. Let L be the list relation $(\text{it}, \text{fr}, \text{jp}, \text{it}, \text{fr}, \text{jp})$, and consider POSS for Q, Γ and L.

It is the case that $L \in Q(\Gamma)$, as shown by the linear extension $id_{42} <' id_{13} <' id_{102} <' id_{20} <' id_{37} <' id_{100}$ of $<$. However, this is hard to see, because tuple values are ambiguous.

Our definitions of the POSS and CERT problems follow the standard notion of *instance* possibility and certainty [4]. Remember that the problems must focus on the uncertainty of *order* (or accumulation results for $\text{PosRA}^{\text{acc}}$), as the underlying relation of PosRA queries is always certain. However, there are other sensible definitions of POSS and CERT for PosRA in our setting, e.g.:

▶ **Definition 13.** The *position possibility* problem asks, given a po-database D, PosRA query Q, tuple t, and rank $k \in \mathbb{N}$, whether $Q(D)$ has a possible world where a tuple with value t occurs at position k. The *position certainty* problem asks whether this is certain.

We will also study the position possibility and certainty problem in the sequel (see Theorem 18). However, as the following example illustrates, we can capture these problems, as well as other variants, with our notion of POSS and CERT for $\text{PosRA}^{\text{acc}}$ queries:

▶ **Example 14.** The position possibility and certainty problems can be reduced to our POSS and CERT problems using the $\text{PosRA}^{\text{acc}}$ query $Q' := \text{select-at-}k(Q)$ (see end of Example 10). Similarly, we can use a query of the form $Q' = \text{top-}k(Q)$ to determine possibility or certainty of a *list of top-k elements*. Alternatively, using an adequate monoid (see Appendix), we can also check, e.g., for two tuple values t_1 and t_2, whether it is *possible* that the first occurrence of value t_1 precedes all occurrences of value t_2.

5 General Complexity Results

We have defined the PosRA and $\text{PosRA}^{\text{acc}}$ query languages, and 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*, where the query Q is fixed[2] (including, for $\text{PosRA}^{\text{acc}}$, the accumulation map and monoid, which we assumed to be PTIME-evaluable): the input to the problem is the po-database D and candidate possible world L. Our results for Sections 5–7 are summarized in Table 1.

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

▶ **Theorem 15.** *The* POSS *problem is NP-complete for* PosRA *and for* $\text{PosRA}^{\text{acc}}$.

Proof sketch. The hardness proof for PosRA is by a reduction from the UNARY-3-PARTITION problem [21]: given numbers written in unary, determine whether they can

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

Table 1 Summary of complexity results for possibility and certainty

	Query	Restrictions	Input relations	Complexity	
`POSS`	PosRA/PosRA$^{\mathrm{acc}}$	—	arbitrary	NP-c.	(Thm. 15)
`CERT`	PosRA$^{\mathrm{acc}}$	—	arbitrary	coNP-c.	(Thm. 16)
`CERT`	PosRA	—	arbitrary	PTIME	(Thm. 17)
`POSS`	PosRA$_{\mathtt{LEX}}$	—	width $\leqslant k$	PTIME	(Thm. 21)
`POSS`	PosRA$_{\mathtt{DIR}}$	—	totally ordered	NP-c.	(Thm. 22)
`POSS`	PosRA	—	ia-width $\leqslant k$	PTIME	(Thm. 26)
`CERT`	PosRA$^{\mathrm{acc}}$	cancellative	arbitrary	PTIME	(Thm. 28)
both	PosRA$^{\mathrm{acc}}$	finite and rank-invariant	totally ordered	NP-c.	(Thm. 31)
both	PosRA$^{\mathrm{acc}}_{\mathtt{LEX}}$	finite	width $\leqslant k$	PTIME	(Thm. 32)
both	PosRA$^{\mathrm{acc}}$	finite and rank-invariant	ia-width $\leqslant k$	PTIME	(Thm. 33)

be partitioned in triples of a fixed sum. The input po-relation represents the numbers of the instance, and the candidate possible world asks whether we can enumerate sequences of three numbers whose total number of elements is the requested sum. This immediately implies the hardness of PosRA$^{\mathrm{acc}}$, using the identity accumulation. ◀

In fact, as we will later point out, hardness holds even for quite restrictive settings, with more intricate proofs: see Theorems 22 and 31.

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

▶ **Theorem 16.** `CERT` *is coNP-complete for* PosRA$^{\mathrm{acc}}$ *queries.*

Proof sketch. We show this by establishing the hardness of `POSS` for a specific PosRA$^{\mathrm{acc}}$ query Q which ensures that only two possible accumulation results may be obtained, no matter the input po-database, so that `POSS` for Q reduces to the negation of `CERT`. The query Q intuitively tests whether its two input po-relations Γ and Γ' have some common possible world, by testing whether there is a possible world enumerating identical elements in alternation from Γ and from Γ'. This is checked by performing accumulation in the transition monoid of a specific deterministic finite automaton. ◀

For PosRA queries, however, we show that `CERT` is in PTIME. This follows from the tractability of `CERT` for PosRA$^{\mathrm{acc}}$ on *cancellative monoids* (Theorem 28).

▶ **Theorem 17.** `CERT` *is in PTIME for* PosRA *queries.*

Other definitions. We can also show that the position possibility and position certainty problems for PosRA (Definition 13) are in PTIME:

▶ **Theorem 18.** *The position possibility and position certainty problems are in PTIME.*

Further tractable cases. We have shown hardness for `POSS` with and without accumulation, and hardness for `CERT` with accumulation. In the next two sections, we identify additional restricted yet realistic cases for which `POSS` and `CERT` become tractable. Section 6 focuses on PosRA (where `CERT` is always tractable) and identifies tractable cases for `POSS`, by restricting the operators allowed, and the "uncertainty" of the input po-relations. Section 7 then shows further tractable cases for `POSS` and `CERT` for PosRA$^{\mathrm{acc}}$ queries.

6 Tractable Cases for `POSS` on PosRA

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

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

Totally Ordered Inputs. We start by the natural case where the individual relations are *totally ordered*. This applies, e.g., to a context where we integrate data from multiple sources, each source being certain (totally ordered), and where uncertainty only results from the integration query. The result of a PosRA query on totally ordered relations is not totally ordered, though, and may still have exponentially many possible worlds (e.g., the *union* of two total orders has exponentially many possible interleavings). The worst offender in this respect is the $\times_{\mathtt{DIR}}$ operator, whose result on two total orders may be arbitrarily "complex". We therefore consider the fragment $\mathrm{PosRA}_{\mathtt{LEX}}$ of PosRA queries without $\times_{\mathtt{DIR}}$, and show:

▶ **Theorem 19.** `POSS` *is PTIME for* $\mathrm{PosRA}_{\mathtt{LEX}}$ *queries if input po-relations are totally ordered.*

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

▶ **Definition 20.** [38] An *antichain* in a po-relation $\Gamma = (ID, T, <)$ is a set $A \subseteq ID$ of pairwise incomparable tuple identifiers. The *width* of Γ is the size of its largest antichain. The *width* of a po-database is the maximal width of its po-relations.

In particular, totally ordered relations have width 1, and unordered relations have a width equal to their size (number of tuples); the width of a po-relation can be computed in PTIME [20]. Po-relations of low width are a common practical case: they cover, for instance, po-relations that are totally ordered except for a few tied tuples at each level. We show:

▶ **Theorem 21.** *Let k be a (constant) positive integer. If the input po-database is of width bounded by k, then* `POSS` *is in PTIME for* $\mathrm{PosRA}_{\mathtt{LEX}}$ *queries.*

Proof sketch. We show that the result Γ with $pw(\Gamma) = Q(D)$ of evaluating the query has bounded width (as $\times_{\mathtt{DIR}}$ is disallowed), and compute in PTIME a *chain partition* of Γ [14, 20] to apply a dynamic algorithm whose state is the position on the chains. ◀

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

▶ **Theorem 22.** *The* `POSS` *problem is NP-complete for* $\mathrm{PosRA}_{\mathtt{DIR}}$ *queries, even when the input po-database is restricted to consist only of totally ordered po-relations.*

Proof sketch. We take the product $R \times_{\mathtt{DIR}} S$ of two totally ordered relations, yielding a grid, and adapt the UNARY-3-PARTITION argument of Theorem 15 to the large antichain on the diagonal, eliminating the rest of the product (see Appendix for the technical argument). ◀

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

▶ **Theorem 23.** `POSS` *is in PTIME for* PosRA *queries if input po-relations are unordered.*

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

▶ **Definition 24.** Given a poset $(V, <)$, a subset $S \subseteq V$ is an *indistinguishable antichain* if it is both an antichain (there are no $x, y \in S$ such that $x < y$) and an *indistinguishable set* (or *interval* [19]): for all $x, y \in S$ and $z \in V \backslash S$, $x < z$ iff $y < z$, and $z < x$ iff $z < y$.

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

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

▶ **Proposition 25.** *The ia-width of any poset, and a corresponding ia-partition, can be computed in PTIME.*

▶ **Theorem 26.** *For any $k \in \mathbb{N}$,* `POSS` *is in PTIME for* PosRA *queries assuming that input po-databases have ia-width $\leqslant k$.*

Proof sketch. As in the proof of Theorem 21, we first show that the query result Γ also has bounded ia-width. We then consider the order on ia-partition classes of Γ. For each linear extension, we apply a greedy algorithm for possibility by mapping candidate tuples to the first available class in the extension where a suitable tuple remains. ◀

7 Tractable Cases for PosRA$^{\text{acc}}$

The previous section illustrated tractable cases for `POSS` on PosRA queries. We now study tractable cases for `POSS` and `CERT` on PosRA$^{\text{acc}}$. In addition to restrictions on the PosRA operators and input po-relations, we will also need to impose restrictions on accumulation (in addition to PTIME-evaluability). Recall that if the monoid is commutative, the result of accumulation is always certain, and therefore `POSS` and `CERT` are trivially in PTIME.

We first start with an approach that only restricts the accumulation operator, from monoids to *cancellative* monoids. We show that `CERT` is tractable for PosRA$^{\text{acc}}$ queries in cancellative monoids, generalizing the tractability of `CERT` for PosRA (Theorem 17); by contrast, `POSS` remains intractable. We then impose other conditions on accumulation (finiteness, and rank-invariance), which allow us to extend the results of Section 6 to PosRA$^{\text{acc}}$.

Cancellative Monoids. We will study accumulation in *cancellative monoids*:

▶ **Definition 27.** [23] For any monoid $(\mathcal{M}, \oplus, \varepsilon)$, we call $a \in \mathcal{M}$ *cancellable* if, for all $b, c \in \mathcal{M}$, we have that $a \oplus b = a \oplus c$ implies $b = c$, and we also have that $b \oplus a = c \oplus a$ implies $b = c$. We call $\mathcal{M}$ a *cancellative monoid* if all its elements are cancellable.

Many interesting monoids are cancellative; in particular, this is the case of all monoids in Example 10. More generally, all *groups* are cancellative monoids (but some infinite cancellative monoids are not groups, e.g., the monoid of concatenation). For this large class of accumulation functions, we design an efficient algorithm for certainty.

▶ **Theorem 28.** `CERT` *is in PTIME for* PosRA$^{\text{acc}}$ *with accumulation in a cancellative monoid.*

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

Hence, `CERT` is tractable for PosRA (Theorem 17), via the concatenation monoid, and `CERT` is also tractable for top-k (defined in Example 10). The hardness of `POSS` for PosRA (Theorem 15) then implies that `POSS`, unlike `CERT`, is hard even on cancellative monoids.

Other Restrictions on Accumulation. We next revisit the results of Section 6 for queries with (PTIME-evaluable) accumulation. However, we first need to introduce other assumptions on accumulation. First, in *all* the following results, we assume that accumulation takes place in a *finite* monoid:

▶ **Definition 29.** A $\mathrm{PosRA}^{\mathrm{acc}}$ query is said to perform *finite* accumulation if the accumulation monoid $(\mathcal{D}', \oplus, \varepsilon)$ is finite.

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

Furthermore, for some results, we will require *rank-invariant accumulation*, namely, that the accumulation map does not depend on the absolute rank of tuples:

▶ **Definition 30.** Recall that the accumulation map h has in general two inputs: a tuple and its rank. A $\mathrm{PosRA}^{\mathrm{acc}}$ query is said to be *rank-invariant* if its accumulation map ignores the second input, so that effectively its only input is the tuple itself.

Note that the monoid operation still receives the input in order, so order-aware accumulation (e.g., concatenation) can still be implemented. We will use these restrictions to lift the results of Section 6. However, note that they do not suffice to make `POSS` and `CERT` tractable:

▶ **Theorem 31.** *`POSS` and `CERT` are respectively NP-hard and coNP-hard for $\mathrm{PosRA}^{\mathrm{acc}}$ queries performing finite and rank-invariant accumulation, even assuming that the input po-database contains only totally ordered po-relations.*

Revisiting Section 6. We now revisit our previous results for queries with accumulation, and for `POSS` and `CERT`, under the additional assumptions on accumulation that we presented. We call $\mathrm{PosRA}^{\mathrm{acc}}_{\mathtt{LEX}}$ the extension of $\mathrm{PosRA}_{\mathtt{LEX}}$ with accumulation.

We can first generalize Theorem 21 to $\mathrm{PosRA}^{\mathrm{acc}}_{\mathtt{LEX}}$ queries with *finite* accumulation:

▶ **Theorem 32.** *For $\mathrm{PosRA}^{\mathrm{acc}}_{\mathtt{LEX}}$ queries performing* finite *accumulation, `POSS` and `CERT` are in PTIME on po-databases whose po-relations have bounded width.*

We can then generalize Theorem 26 to $\mathrm{PosRA}^{\mathrm{acc}}$ queries, assuming *finite* and *rank-invariant* accumulation:

▶ **Theorem 33.** *For $\mathrm{PosRA}^{\mathrm{acc}}$ queries performing* finite *and* rank-invariant *accumulation, `POSS` and `CERT` are in PTIME on po-databases whose po-relations have bounded ia-width.*

The finiteness assumption is important, as the previous result does not hold otherwise. Specifically, we can show a query that performs *rank-invariant* but not *finite* accumulation, for which `POSS` is NP-hard even on unordered po-relations (see Appendix).

8 Duplicate Consolidation

We last study the problem of consolidating tuples with *duplicate values*. We have only considered *bag* semantics for PosRA so far, but in some cases users may wish to treat duplicate tuples as if they refer to the same object, and choose to collapse different occurrences into a single tuple, without relying on rank aggregation techniques to decide on a particular order.

Thus, 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. Hence, the representative tuples that we keep may yield different orders on the result, i.e., introduce more *order uncertainty*. To mitigate this, we introduce the notion of *id-sets*:

▶ **Definition 34.** Given a list relation $L = (t_1, ..., t_n)$, a subset S of the tuples in L is an *indistinguishable duplicate set* (or *id-set*) if for every $t_i, t_j \in S$, we have $t_i = t_j$, and for every $t \in L \backslash S$, we have that t precedes (resp. follows) t_i in L iff t precedes (resp. follows) t_j in L.

▶ **Example 35.** Consider the list relation defined by $L_1 := \Pi_{hotelname}(Hotel)$, 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 list relation $L_2 := (A, B, B, C)$, where A, B, and C are tuples over $\mathcal{D}$. The two occurrences of B form an id-set. Note that a singleton is always an id-set.

We define a semantics for dupElim on any list relation L using id-sets. First, check that for every tuple t in L, the occurrences of t form an id-set. If this holds, we say that L is *safe*, and we set $\text{dupElim}(L)$ to be the single possible world obtained by picking one representative element per id-set (clearly the result does not depend on the chosen representatives). Otherwise, we call L *unsafe* and say that duplicate consolidation has *failed*; we set $\text{dupElim}(L)$ to be an empty set of possible worlds. Intuitively, duplicate consolidation tries to reconcile (or "synchronize") order constraints for tuples sharing the same values, and fails when this cannot be done. We discuss other possibilities at the end of this section.

▶ **Example 36.** In Example 35, we have $\text{dupElim}(L_1) = \emptyset$ but $\text{dupElim}(L_2) = (A, B, C)$.

We then extend the semantics of dupElim to po-relations. We consider all possible results of duplicate elimination on the possible worlds, ignoring the unsafe possible worlds. If all possible worlds are unsafe, then we *completely fail*.

▶ **Definition 37.** Letting Γ be a po-relation, we define $\text{dupElim}(\Gamma) := \bigcup_{L \in pw(\Gamma)} \text{dupElim}(L)$. $\text{dupElim}(\Gamma)$ *completely fails* if $\text{dupElim}(\Gamma) = \emptyset$, that is, $\text{dupElim}(L) = \emptyset$ for every $L \in pw(\Gamma)$.

▶ **Example 38.** Consider the totally ordered relation $Rest_3 :=$ (Tsukizi, Gagnaire) and *Rest* as in Figure 1, and the query $Q := \text{dupElim}(\Pi_{restname}(Rest) \cup Rest_3)$. Intuitively, Q combines restaurant rankings, performing duplicate consolidation to collapse two occurrences of the same restaurant name into a single tuple. The only possible world of Q is (Tsukizi, Gagnaire, TourArgent), since duplicate elimination fails in the other possible worlds of the union, and this is indeed the only possible way to combine the rankings.

We next show that po-relations still form a strong representation system for PosRA with dupElim, up to complete failure (which may be efficiently identified).

▶ **Theorem 39.** *For any po-relation* Γ*, we can test in PTIME if* $\text{dupElim}(\Gamma)$ *completely fails; if it does not, we can compute in PTIME a po-relation* Γ' *such that* $pw(\Gamma') = \text{dupElim}(\Gamma)$.

Possibility and certainty. All complexity results of Sections 5–7 continue to hold when extending PosRA and $\text{PosRA}^{\text{acc}}$ to allow dupElim. To prove this, we use Theorem 39, and show that the width and ia-width order complexity bounds of Section 6 are also preserved by dupElim (see Appendix for formal result and proof). Furthermore, if in a set-semantics spirit we *require* that the query output has no duplicates, `POSS` and `CERT` are always tractable:

▶ **Theorem 40.** *For any* PosRA *query* Q*,* `POSS` *and* `CERT` *for* $\text{dupElim}(Q)$ *are in PTIME.*

Alternative semantics. A main downside of our proposed semantics for dupElim is the fact that complete failure is allowed. We conclude this section by briefly considering alternative semantics that avoid failure, and illustrate the other problems that they have.

A first possibility is to do a *weak* form of duplicate elimination: keep one element for each *maximal id-set*, rather than for each value, and leave some duplicates in the output:

▶ **Example 41.** Letting $A \neq B$ be two tuples, let us consider the totally ordered relation $L := (A, B, B, A)$. With weak duplicate elimination, we would have $\text{dupElim}(L) = (A, B, A)$.

However, when generalizing this semantics from totally ordered relations to po-relations, we notice that the result of dupElim on a po-relation may not be representable as a po-relation, since possible worlds differ in their tuples and not only on their order:

▶ **Example 42.** Consider the po-relation $\Gamma = (\{a_1, b, a_2\}, T, <)$ with $T(a_1) = T(a_2) = A$ and $T(b) = B$, where $A \neq B$ are tuples, and $<$ defined by $a_1 < b$ and $a_1 < a_2$. We have $pw(\Gamma) = \{(A, B, A), (A, A, B)\}$ and $\text{dupElim}(\Gamma) = \{(A, B, A), (A, B)\}$ for *weak* duplicate elimination: we cannot represent it as a po-relation (the underlying relation is not certain).

A second possibility is to do an *aggressive* form of duplicate elimination: define dupElim(L) for totally ordered L as the set of *all* totally ordered relations that we can obtain by picking one representative element for each value, even when the representatives are not indistinguishable. In other words, we do not fail even if we cannot reconcile the order between duplicate tuples:

▶ **Example 43.** Applying *aggressive* dupElim to Γ from Example 41 yields $\{(A, B), (B, A)\}$.

However, again dupElim(Γ) may not be representable as a po-relation, this time because the set of possible orders may not correspond to a partial order:

▶ **Example 44.** Consider $L := (A, C, B, C, A)$ with distinct tuples A, B, C. Then dupElim(L) is $\{(A, C, B), (A, B, C), (B, C, A), (C, B, A)\}$. No po-relation Γ satisfies $pw(\Gamma) = \text{dupElim}(L)$, because no comparability pair holds in all possible worlds, so Γ must be unordered, but then all permutations of $\{A, B, C\}$ are possible worlds of Γ, which is unsuitable.

We leave for future work the question of designing a practical semantics for duplicate consolidation that maintains an efficient representation system while avoiding failure.

9 Related Work

Incompleteness in databases. Incomplete information management has been studied for various models [6, 30], in particular relational databases [24]. This field inspires our design of po-relations as a strong representation system, and our study of possibility and certainty [4, 34]. However, uncertainty in these settings typically focuses on *whether* tuples exist or on what their *values* are (e.g., with nulls [11], including the novel approach of [31, 32]; with c-tables [24], probabilistic databases [44] or fuzzy numerical values as in [40]).

To our knowledge, though, our work is the first to study possible and certain answers in the general context of *order*-incomplete data (see discussion below of uncertain order in different contexts). Combining order incompleteness with standard tuple-level uncertainty is left as a challenge for future work. Note that some works on incomplete databases [9, 29, 32] use partial orders on *relations* to compare the informativeness of uncertain representations. However, this is unrelated to our use of partial orders on *tuples* as a representation system.

Trees, bags, lists, posets, and pomsets. Our work focuses on querying *ordered relations*, with uncertainty with respect to order. Expressive query languages have been designed for *bags* [33] and for ordered structures such as *lists* [12, 13] and *trees* [37], usually extending the relational algebra to the *nested* relational algebra [33]. However, these works often do not handle uncertainty, and thus do not address the problems that we study here.

Uncertainty with respect to order is of course well-studied in the context of order theory. In particular, *labeled partial orders* [36] are essentially equivalent to our po-relations, with "labels" corresponding to tuples. However, we are unaware of works on labeled partial orders that investigate query languages over them or complexity issues, to the notable exception of [22], which studies an algebra for *pomsets*. Our approach and results are different, however: we focus on the investigation of `POSS` and `CERT`, which [22] does not study; in fact, as [22] allows a very expressive language, our complexity results would probably fail in their setting.

Ordered domains. Another line of work has studied relational data management where the *domain elements* are ordered, rather than the tuples: some works assume a total order, hence no uncertainty [25], but others assume a partial order [35, 45]. However, the perspective is different: we see order on tuples as part of the relations, and as being constructed by applying our operators; these works see order on elements as being given *outside* of the query. Hence, unlike us, they do not study how uncertainty is propagated and generated while evaluating queries. Last, queries in such works can often directly access the order relation on the domain [45, 7], which impacts their complexity results.

Some works also investigate the possible orders that can be expressed via numerical uncertainty on totally ordered *numerical* domains [40, 41], whereas we look at general order relations. In this context, some of the present authors are submitting another work to the same venue [3]; the problem studied there is very different, however, as it focuses on probabilities and unknown numerical values under order constraints on the values, and on top-k computation, rather than our query language and the problems of `POSS` and `CERT`.

Practical Implementations. Uncertain order on tuples arises in the context of many practical systems. For instance, unioning two sorted relations in SQL implementations yields an ordered bag relation: the order is implementation-dependent, but there is no representation of the multiple possibilities. Indeed, by the SQL standard, "ordering of the rows of the table specified by the query expression is guaranteed only for the query expression that immediately contains the `ORDER BY` clause" [26]. SQL also rejects some queries that combine `DISTINCT` with `ORDER BY`. Query languages for XML follow a similar approach: see, e.g., Section 3.4.2 in [46]. Our work can thus be seen as a *generic attempt* to fill these gaps.

Temporal Databases. *Temporal databases* [10, 39] consider order on facts, but it is usually induced by timestamps, hence total. A notable exception is [18] 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 [27, 5, 28, 2, 43], with some operators being the same as ours, and for *rank aggregation* [17, 27, 16], the problem of retrieving top-k query answers given possibly incompatible 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 [17], or a preference function [2]). The problems that we study are complementary: we focus on the querying of uncertain data in a compositional way, namely, maintaining a faithful model of *all* possible worlds without assuming or making any intermediate choice on how to reconcile them; we then return possible and certain answers with respect to all possible worlds.

10 Conclusion

This paper introduced an algebra for order-incomplete data, based on the bag semantics of the positive relational algebra, and proposed an order-aware accumulation operator. We have studied the complexity of possible and certain answers for this algebra, including duplicate consolidation. We have shown that the problems are generally intractable, but identified useful tractable cases by limiting the query language, accumulation operator, and input data.

An important direction for future work is to add other operators (e.g., group-by, list map, difference, and others from [22]) and study the impact on our results. Other directions include the search for different semantics, e.g., for duplicate elimination, and the investigation of how to combine order-uncertainty with uncertainty on values (e.g., `NULL`s).

REFERENCES

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2 B. Alexe, M. Roth, and W.-C. Tan. Preference-aware integration of temporal data. *PVLDB*, 8(4):365–376, Dec. 2014.
3 A. Amarilli, Y. Amsterdamer, T. Milo, and P. Senellart. Top-k queries on unknown values under order constraints. `http://pierre.senellart.com/publications/amarilli2016top.pdf`, 2016. Submitted for publication.
4 L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *ICDT*. 2007.
5 A. Arvanitis and G. Koutrika. PrefDB: Supporting Preferences as First-Class Citizens in Relational Databases. *IEEE TKDE*, 26(6), 2014.
6 P. Barceló, L. Libkin, A. Poggi, and C. Sirangelo. XML with incomplete information. *J. ACM*, 58(1), 2010.
7 M. Benedikt and L. Segoufin. Towards a characterization of order-invariant queries over tame graphs. *The Journal of Symbolic Logic*, 74:168–186, 3 2009.
8 A. Brandstädt, V. B. Le, and J. P. Spinrad. Posets. In *Graph Classes. A Survey*, chapter 6. SIAM, 1987.
9 P. Buneman, A. Jung, and A. Ohori. Using powerdomains to generalize relational databases. *Th. Comp. Sci.*, 91(1), 1991.
10 J. Chomicki and D. Toman. Time in database systems. In *Handbook of Temporal Reasoning in Artificial Intelligence*. Elsevier, 2005.
11 E. F. Codd. Extending the database relational model to capture more meaning. *TODS*, 4(4), 1979.
12 L. S. Colby, E. L. Robertson, L. V. Saxton, and D. V. Gucht. A query language for list-based complex objects. In *PODS*, 1994.
13 L. S. Colby, L. V. Saxton, and D. V. Gucht. Concepts for modeling and querying list-structured data. *Inf. Process. Manage.*, 30(5), 1994.
14 R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 1950.
15 B. Dushnik and E. W. Miller. Partially ordered sets. *Amer. J. Math.*, 63(3), 1941.
16 C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*. ACM, 2001.
17 R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
18 W. Fan, F. Geerts, and J. Wijsen. Determining the currency of data. *TODS*, 37(4), 2012.
19 R. Fraïssé. L'intervalle en théorie des relations; ses genéralisations, filtre intervallaire et clôture d'une relation. *North-Holland Math. Stud.*, 99, 1984.
20 D. R. Fulkerson. Note on dilworth's decomposition theorem for partially ordered sets. In *Proc. Amer. Math. Soc*, 1955.
21 M. R. Garey and D. S. Johnson. *Computers And Intractability. A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.
22 S. Grumbach and T. Milo. An algebra for pomsets. In *ICDT*, 1995.
23 J. M. Howie. *Fundamentals of semigroup theory*. Oxford: Clarendon Press, 1995.
24 T. Imieliński and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.
25 N. Immerman. Relational queries computable in polynomial time. *Inf. Control*, 68(1-3), 1986.
26 ISO. ISO 9075:2008: SQL, 2008.
27 M. Jacob, B. Kimelfeld, and J. Stoyanovich. A system for management and analysis of preference data. *VLDB Endow.*, 7(12), 2014.

28 W. Kiessling. Foundations of preferences in database systems. In *VLDB*, 2002.
29 L. Libkin. A semantics-based approach to design of query languages for partial information. In *Semantics in Databases*, 1998.
30 L. Libkin. Data exchange and incomplete information. In *PODS*, 2006.
31 L. Libkin. Incomplete data: What went wrong, and how to fix it. In *PODS*, 2014.
32 L. Libkin. SQL's three-valued logic and certain answers. In *ICDT*, 2015.
33 L. Libkin and L. Wong. Query languages for bags and aggregate functions. *J. Comput. Syst. Sci.*, 55(2), Oct. 1997.
34 W. Lipski, Jr. On semantic issues connected with incomplete information databases. *TODS*, 4(3), Sept. 1979.
35 W. Ng. An extension of the relational data model to incorporate ordered domains. *TODS*, 26(3), 2001.
36 V. R. Pratt. The pomset model of parallel processes: Unifying the temporal and the spatial. In *Seminar on Concurrency*, 1984.
37 E. L. Robertson, L. V. Saxton, D. Van Gucht, and S. Vansummeren. Structural recursion as a query language on lists and ordered trees. *TCS*, 44(4), 2009.
38 B. Schröder. *Ordered Sets: An Introduction*. Birkhäuser, 2003.
39 R. T. Snodgrass, J. Gray, and J. Melton. *Developing time-oriented database applications in SQL*. Morgan Kaufmann, 2000.
40 M. A. Soliman and I. F. Ilyas. Ranking with uncertain scores. In *ICDE*, 2009.
41 M. A. Soliman, I. F. Ilyas, and S. Ben-David. Supporting ranking queries on uncertain and incomplete data. *VLDBJ*, 19(4), 2010.
42 R. P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 1986.
43 K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *TODS*, 36(3), Aug. 2011.
44 D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
45 R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *JCSS*, 54(1), 1997.
46 W3C. XQuery 3.0: An XML query language, 2014.

A Proofs for Section 2 (Data Model and PosRA)

A.1 Additional Preliminaries

A *bag* or *multiset* over a set X is a function $B : X \to \mathbb{N}$. The *support* of a bag B is $B^{-1}(\mathbb{N}^+)$ and we write $x \in B$ if and only if $B(x) \neq 0$. We write a bag B with finite support as $B = \{\{ b_1, \ldots, b_n \}\}$ where $n = \sum_{x \in X} B(x)$ is the *size* $|B|$ of B: for every $x \in X$, we have $B(x) = |\{1 \leqslant k \leqslant n \mid b_k = x\}|$. For any bag B and Boolean predicate φ on elements of X, the bag $\{\{ x \in B \mid \varphi(x) \}\}$ is the function that maps x to $B(x)$ if $\varphi(x)$ holds and maps x to 0 otherwise.

For any two bags B_1, B_2 over the same set X, $B_1 \uplus B_2$ is the bag over X defined by $x \mapsto B_1(x) + B_2(x)$. For any bag B over X and any function F from X to bags over X, $\biguplus_{x \in B} F(x)$ is the bag over X defined by $y \mapsto \sum_{x \in B} B(x) \cdot F(x)(y)$.

We define the *Boolean formulas over tuples* that will be used for the selection operator – for simplicity, we sometimes adopt in the proofs the unnamed perspective and thus identify positions within tuples by their index:

▶ **Definition 45.** A *tuple predicate* is a Boolean formula over atoms of the form "$.m = .n$", "$.m \neq .n$", "$.m = d$", or "$.m \neq d$" where m, n are positive integers and $d \in \mathcal{D}$.

A tuple predicate φ of the form "$.m = .n$" (resp., "$.m \neq .n$") *holds for a tuple* t, denoted $\varphi(t)$, if and only if $m \leqslant \mathrm{a}(t)$, $n \leqslant \mathrm{a}(t)$, and $t.m = t.n$ (resp., $t.m \neq t.n$). A tuple predicate φ of the form "$.m = d$" (resp. "$.m \neq d$") *holds for a tuple* t, denoted $\varphi(t)$, if and only if $m \leqslant \mathrm{a}(t)$ and $t.m = d$ (resp., $t.m \neq d$).

Given a totally ordered relation $L = (t_1, \ldots, t_n)$, for two tuples t_i and t_j of L, we write $t_i \leqslant_L t_j$ (resp. $t_i <_L t_j$) to mean that t_i precedes (resp. strictly precedes) t_j, i.e., $i \leqslant j$ (resp. $i < j$).

A.2 Proof of Theorem 4

▶ **Theorem 4.** *No* PosRA *operator can be expressed through a combination of the others.*

We prove Theorem 4 by considering each operator in turn, showing it cannot be expressed through a combination of the others.

We first consider constant expressions. We will 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 $\mathbb{N}^*_{\leqslant n}$ constant expression, and so this query has no equivalent that does not use the $[t]$ constant expression.
- For $\mathbb{N}^*_{\leqslant n}$, observe that $\mathbb{N}^*_{\leqslant 2}$ 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, the first three are easily shown to be non-expressible:

- σ is the only operator that can decrease the size of an input po-relation.
- Π is the only operator that can decrease the arity of an input po-relation.
- $[\langle 0 \rangle] \cup [\langle 1 \rangle]$ (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 in the same attribute the two elements 0 and 1.

There remains to prove that $\times_{\mathtt{DIR}}$ and $\times_{\mathtt{LEX}}$ are not redundant. As in Section 6, we use the name $\mathrm{PosRA}_{\mathtt{DIR}}$ for the fragment of PosRA where $\times_{\mathtt{LEX}}$ is not used; and $\mathrm{PosRA}_{\mathtt{LEX}}$ for the fragment of PosRA where $\times_{\mathtt{DIR}}$.

A.2.1 Transformations Not Expressible in $\mathrm{PosRA}_{\texttt{LEX}}$

Let us start by showing that $\mathrm{PosRA}_{\texttt{DIR}}$ can express some transformations that $\mathrm{PosRA}_{\texttt{LEX}}$ cannot: specifically, the output of a $\mathrm{PosRA}_{\texttt{LEX}}$ query is always a *series-parallel po-relation* when the input relations also are.

▶ **Definition 46.** [Sch03] The *series-parallel (sp) posets* is the class of posets containing all single-element posets and defined inductively as follows: for any two sp-posets $P_1 = (V_1, <_1)$ and $P_2 = (V_2, <_2)$ with disjoint domains, we can build the following sp-posets on $V_1 \sqcup V_2$, whose orders follow $<_1$ (resp. $<_2$) on $V_1 \times V_1$ (resp. $V_2 \times V_2$):

Series composition: set $p < p'$ for any $(p, p') \in V_1 \times V_2$;
Parallel composition: make p and p' are incomparable for any $(p, p') \in V_1 \times V_2$.

A *series-parallel po-relation* is a po-relation whose underlying poset is either sp or empty.

We first introduce the notion of *sp-tree* to make it easier to reason about series-parallel posets:

▶ **Definition 47.** An *sp-tree* [Bv96] is a rooted ordered tree whose internal nodes are labeled either series or parallel, and leaf nodes are labeled with singleton. The *decoding* of an sp-tree is a series-parallel poset (defined up to isomorphism) obtained in the following fashion:

- the decoding of a singleton node is the poset $(\{s\}, \emptyset)$ where s is a fresh element;
- the decoding of a series node is the series composition of the posets obtained as the decoding of the children of this node, in the order in which they appear;
- likewise, the decoding of a parallel node is the parallel composition of the decoding of the children.

We now show $\mathrm{PosRA}_{\texttt{LEX}}$ queries preserve being series-parallel.

▶ **Proposition 48.** *Let Q be any $\mathrm{PosRA}_{\texttt{LEX}}$ and D a po-database D whose po-relations are all series-parallel. Then for any po-relation Γ such that $pw(\Gamma) = Q(D)$, Γ is series-parallel.*

Proof. We prove the claim by induction. For the base case:

- The relations of D are series-parallel.
- The expressions $[t]$ and $\mathbb{N}^*_{\leqslant n}$ result in series-parallel orders.

For the induction step:

- The union of two series-parallel po-relations of compatible arity is a series-parallel po-relation, whose underlying poset is the parallel composition of the two original posets.
- The projection of a series-parallel po-relation is still series-parallel (the underlying poset does not change).
- The selection of a series-parallel po-relation has an underlying poset which is either empty or is a non-empty restriction of a series-parallel poset, so it is still series-parallel [BGR97].
- The LEX product $R'' := R \times_{\texttt{LEX}} R'$ of two series-parallel relations R and R' is series-parallel. To show this, note that If either of R or R' are empty, then the product is also empty. Otherwise, the underlying poset P'' of R'' is defined as the lexicographic product of the underlying posets P and P' of R and R' respectively, which are series-parallel. To see why P'' is series-parallel, consider any sp-trees T and T' of P and P' respectively. Clearly, the result of replacing every singleton node of T by a copy of T' is an sp-tree for P. Hence, P'' is series-parallel.

This concludes the proof. ◀

This allows us to conclude:

▶ **Corollary 49.** *There are transformations expressible in* $\mathrm{PosRA}_{\mathtt{DIR}}$ *but not in* $\mathrm{PosRA}_{\mathtt{LEX}}$.

Proof. By Proposition 48, any transformation expressed by a $\mathrm{PosRA}_{\mathtt{LEX}}$ query is such that the image of a po-database of totally ordered relations is a series-parallel po-relation (see Definition 46). Hence, to show that some transformations can be expressed by $\mathrm{PosRA}_{\mathtt{DIR}}$ but not by $\mathrm{PosRA}_{\mathtt{LEX}}$, it suffices to provide an example of a $\mathrm{PosRA}_{\mathtt{DIR}}$ query Q and series-parallel po-database D such that $Q(D)$ is the set of possible worlds of a non–series-parallel po-relation.

Consider Q the query $\sigma_\varphi(\mathbb{N}^*_{\leqslant 2} \times_{\mathtt{DIR}} \mathbb{N}^*_{\leqslant 3})$ and D the empty po-database, where φ is the tuple predicate:

$$(.1 = \text{“2”} \wedge .2 = \text{“1”}) \vee (.1 = \text{“2”} \wedge .2 = \text{“2”}) \vee (.1 = \text{“1”} \wedge .2 = \text{“2”}) \vee (.1 = \text{“1”} \wedge .2 = \text{“3”})$$

It is easily verified that $Q(D)$ is the set of possible worlds of a po-relation Γ with four tuples t_1, t_2, t_3 and t_4, with respective values $\langle 2,1\rangle$, $\langle 2,2\rangle$, $\langle 1,2\rangle$ and $\langle 1,3\rangle$, such that exactly the following comparability relations hold: $t_1 < t_2$, $t_3 < t_2$, $t_3 < t_4$. But this is exactly the N-shaped poset of [Möh89] which is an example of a non-series-parallel poset. Hence, Γ is not series-parallel, proving the desired result. ◀

A.2.2 Transformations Not Expressible in $\mathrm{PosRA}_{\mathtt{DIR}}$

We now show the converse, that $\mathrm{PosRA}_{\mathtt{LEX}}$ expresses some transformations that cannot be expressed in $\mathrm{PosRA}_{\mathtt{DIR}}$. To do this, we introduce *concatenation* as follows:

▶ **Definition 50.** For L_1 and L_2 two list relations with $\mathrm{a}(L_1) = \mathrm{a}(L_2)$, the *concatenation* of L_1 and L_2, written $L_1 \cup_{\mathtt{CAT}} L_2$, is *the* set formed of the single list where all tuples of L_1 (in order) come before those of L_2 (in order).

We extend concatenation to po-relations by defining the result of concatenating two po-relations as series composition of their two partial orders. Its set of possible worlds is the set of all concatenations of a possible world of the first relation and a possible world of the second relation. We show that concatenation can be captured with $\mathrm{PosRA}_{\mathtt{LEX}}$.

▶ **Lemma 51.** *For any arity $n \in \mathbb{N}$, there is a* $\mathrm{PosRA}_{\mathtt{LEX}}$ *query Q_n with two distinguished relation names R and R' 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', $Q_n(D)$ is $pw(\Gamma \cup_{\mathtt{CAT}} \Gamma')$.*

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

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

It is easily verified that Q_n satisfied the claimed property. ◀

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

▶ **Lemma 52.** *For any arity $n \in \mathbb{N}_+$ and distinguished relation names R and R', there is no* $\mathrm{PosRA}_{\mathtt{DIR}}$ *query Q_n such that, for any po-relations Γ and Γ', letting D be the po-database that maps R to Γ and R' to Γ', $Q_n(D)$ evaluates to $pw(\Gamma \cup_{\mathtt{CAT}} \Gamma')$.*

To prove Lemma 52, we first introduce the following concept:

▶ **Definition 53.** Let $v \in \mathcal{D}$. We call a po-relation Γ *v-impartial* if, for any two tuples t_1 and t_2 and $1 \leqslant i \leqslant \mathrm{a}(\Gamma)$ such that exactly one of $t_1.i$, $t_2.i$ is v, the following holds: t_1 and t_2 are *incomparable*, namely, t_1 precedes t_2 in some possible order of Γ, and t_2 precedes t_1 in some possible order of Γ.

▶ **Lemma 54.** *Let $v \in \mathcal{D}\backslash\mathbb{N}$ be a value. For any* $\mathrm{PosRA}_{\mathtt{DIR}}$ *query Q, for any po-database D of v-impartial po-relations, any po-relation Γ such that $pw(\Gamma) = Q(D)$ is v-impartial.*

Proof. Let $v \in \mathcal{D} \backslash \mathbb{N}$ be such a value. We show the claim by induction on the query Q.

The base cases are the following:

- For the base relations, the claim is vacuous by our hypothesis on D.
- For the empty and singleton constant expressions, the claim is trivial as they contain less than two tuples.
- For the $\mathbb{N}^*_{\leqslant i}$ 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 tuples t_1 and t_2 in Γ and $1 \leqslant i \leqslant \mathrm{a}(\Gamma)$ satisfying the condition, as Γ is v-impartial, t_1 and t_2 are incomparable in Γ', so they are also incomparable in Γ: applying the selection to the two possible orders witnessing impartiality in Γ', yields two possible orders of Γ witnessing its v-impartiality.
- For projection, the claim is also immediate as the property to prove is maintained when reordering, copying or deleting attributes. Indeed, considering again two tuples t_1 and t_2 of Γ and $1 \leqslant i \leqslant \mathrm{a}(\Gamma)$, the preimage t'_1 and t'_2 of t_1 and t_2 before the projection satisfy the same condition for some different i' which is the preimage of i, so we again use the impartiality of the original po-relation to conclude.
- For union, the property is preserved. Indeed, for $\Gamma'' = \Gamma \cup \Gamma'$, assume by contradiction the existence of two tuples $t_1, t_2 \in \Gamma''$ and $1 \leqslant i \leqslant \mathrm{a}(\Gamma'')$ such that exactly one of $t_1.i$ and $t_2.i$ is v but (without loss of generality) t_1 precedes t_2 in every possible world of Γ''. It is easily seen that, as t_1 and t_2 are not incomparable, they must come from the same relation; but then, as that relation was v-impartial, we have a contradiction.
- We now show that the property is preserved for $\times_{\mathtt{DIR}}$. Consider $\Gamma'' = \Gamma \times_{\mathtt{DIR}} \Gamma'$ where Γ and Γ' are v-impartial, and assume that there are two tuples $\langle t_1, t_2 \rangle$ and $\langle t'_1, t'_2 \rangle$ in Γ'' and $1 \leqslant i \leqslant \mathrm{a}(\Gamma'')$ that violate the v-impartiality of Γ''. 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 $t_1.i$ and $t'_1.i$ is v, so that in particular $t_1 \neq t'_1$. Thus, by definition of the order in $\times_{\mathtt{DIR}}$, it is easily seen that, because (t_1, t_2) precedes (t'_1, t'_2) in every possible world of Γ'', t_1 must precede t'_1 in every possible world of Γ, contradicting the v-impartiality of Γ. The second case is symmetric. ◀

We now conclude with the proof of Lemma 52:

Proof. Let us assume by way of contradiction that there is $n \in \mathbb{N}_+$ and a $\mathrm{PosRA}_{\mathtt{DIR}}$ query Q_n capturing $\cup_{\mathtt{CAT}}$, with Γ'' a po-relation such that $pw(\Gamma'') = Q_n(D)$. Let $v \neq v'$ be two distinct values in $\mathcal{D} \backslash \mathbb{N}$, consider the singleton po-relations $\Gamma = (t)$ and $\Gamma' = (t')$, 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'. Now, as Γ and Γ' are (vacuously) v-impartial, we know by Lemma 54 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, there is a possible world of Γ'' where t' precedes t. This contradicts the fact that we should have $\Gamma'' = \Gamma \cup_{\mathtt{CAT}} \Gamma'$, namely, $\Gamma'' = (t, t')$, which has a single possible world where t' does not precede t. This proves that $\cup_{\mathtt{CAT}}$ cannot in fact be captured by a $\mathrm{PosRA}_{\mathtt{DIR}}$ query. ◀

Corollary 49, Lemma 51, and Lemma 52 conclude the proof of Theorem 4.

A.3 Proof of Proposition 6

▶ **Proposition 6.** *For any* PosRA *query Q and po-database D, we can compute in polynomial time in D (the exponent depending on Q) a po-relation Γ such that $pw(\Gamma) = Q(D)$.*

Proof. We show the claim by induction on the query Q.

- If Q is a relation name R, $Q(D) = pw(D(R))$, with $D(R)$ obtained in time linear in D.
- If $Q = [t]$, we let Γ be the po-relation on the singleton tuple (t).
- If $Q = \mathbb{N}^*_{\leqslant n}$, we let $\Gamma \coloneqq ([\![1;n]\!], k \mapsto (k), <)$ where $<$ is the total order over integers. This has constant size in D.
- If $Q = \sigma_\varphi(Q')$, writing $Q'(D) = pw(\Gamma')$ with $\Gamma' = (ID', T', <')$ by the induction hypothesis, let ID be the set of all $\iota \in ID'$ such that $\varphi(T'(\iota))$ holds. Then we let $\Gamma \coloneqq (ID, T'_{|ID}, <'_{|ID})$, which is constructible in time linear in Γ'.
- If $Q = \Pi_{k_1\ldots k_p}(Q')$, writing $Q'(D) = pw(\Gamma')$ with $\Gamma' = (ID', T', <')$ by the induction hypothesis, define $T : \iota \mapsto \Pi_{k_1\ldots k_p}(T'(\iota))$. Then we let $\Gamma \coloneqq (ID', T, <')$, which is constructible in time linear in Γ'.
- If $Q = Q_1 \cup Q_2$, for $i \in \{1,2\}$, use the induction hypothesis to write $Q_i(D) = pw(\Gamma_i)$ with $\Gamma_i = (ID_i, T_i, <_i)$. If ID_1 and ID_2 are not disjoint, we rename identifiers from one of them to fresh identifiers, redefining T_i and $<_i$ accordingly, which is linear in D. Hence, we assume without loss of generality that ID_1 and ID_2 are disjoint.
  We let $\Gamma \coloneqq (ID_1 \cup ID_2, T_1 \cup T_2, <_1 \cup <_2)$. This construction is linear in Γ_1 and Γ_2. We will now prove that this gives the right semantics, using the fact that a linear extension of the union of two partial orders on disjoint domains is an arbitrary interleaving of linear extensions of the two partial orders.
  For the forward direction, let L be a possible world of Γ. By our remark above about Γ, there is a possible world L_1 of Γ_1 and L_2 of Γ_2 such that L is an interleaving of L_1 and L_2. By the induction hypothesis, we have $L_1 \in Q_1(D)$ and $L_2 \in Q_2(D)$. Since $(Q_1 \cup Q_2)(D)$ is formed of all interleavings of $Q_1(D)$ and $Q_2(D)$, we have $L \in (Q_1 \cup Q_2)(D) = Q(D)$.
  For the backward direction, let $L \in Q(D)$. By our remark above, L is an interleaving of a $L_1 \in Q_1(D)$ and a $L_2 \in Q_2(D)$. By the induction hypothesis, we have $L_1 \in pw(\Gamma_1)$ and $L_2 \in pw(\Gamma_2)$. Thus, L is a possible world of Γ.
- If $Q = Q_1 \times_{\texttt{DIR}} Q_2$, for $i \in \{1,2\}$, use the induction hypothesis to write $Q_i(D) = pw(\Gamma_i)$ with $\Gamma_i = (ID_i, T_i, <_i)$. We define $\Gamma \coloneqq (ID_1 \times ID_2, T, <)$ where $T : (\iota_1, \iota_2) \mapsto \langle T(\iota_1), T(\iota_2)\rangle$ and $<$ is defined as the minimal order relation such that $(\iota_1, \iota_2) < (\iota'_1, \iota'_2)$ whenever there are $i \neq j \in \{1,2\}$ such that $\iota_i < \iota'_i$ and $\iota_j \leqslant \iota'_j$ (i.e., either $\iota_j = \iota'_j$ or $\iota_j < \iota'_j$). We can construct this in time polynomial in the product of the size of Γ_1 and Γ_2, hence, in time polynomial in D: to construct the order, enumerate all pairs that are as above, and then complete the set of constraints into an order in PTIME via transitive closure.
  Now, to prove correctness, let L be a possible world of Γ. The definition of $<$ ensures there is no $(\iota_1, \iota_2) < (\iota'_1, \iota'_2)$ if $\iota'_i \leqslant \iota_i$ for all $i \in \{1,2\}$. This means $L \in Q(D)$. Conversely, if $L \in Q(D)$, L does not violate any of the constraints of $<$, and is therefore a possible world of Γ.
- If $Q = Q_1 \times_{\texttt{LEX}} Q_2$, for $i \in \{1,2\}$, use the induction hypothesis to write $Q_i(D) = pw(\Gamma_i)$ with $\Gamma_i = (ID_i, T_i, <_i)$. We define $\Gamma \coloneqq (ID_1 \times ID_2, T, <)$ where T is as in the previous case and $<$ is the lexicographic product of the orders $<_1$ and $<_2$. This is constructible in linear time in the size of the product of Γ_1 and Γ_2, and the definition of $\times_{\texttt{LEX}}$ ensures that possible worlds of Γ are exactly possible outcomes of Q over $pw(D)$. ◀

B Proofs for Section 4 (Possibility and Certainty)

Capturing the possibility of order relations. To check whether the first occurrence of a tuple t_1 precedes any occurrence of t_2, we use the accumulation operator $\mathrm{accum}_{h,\oplus}$ defined as follows. We define the accumulation map h by $h(t_1, n) = \top$, $h(t_2, n) = \bot$ and $h(t, n) = \varepsilon$ for $t \neq t_1, t_2$. We define the monoid operator $\oplus$ by imposing $\top \oplus \top = \top \oplus \bot = \top$ and $\bot \oplus \bot = \bot \oplus \top = \bot$. This ensures that evaluating $\mathrm{accum}_{h,\oplus}(L)$ on a totally ordered relation

L yields ε if neither t_1 not t_2 is present, $\top$ if the first occurrence of t_1 precedes any occurrence of t_2, and $\perp$ otherwise. Hence, to check whether it is possible that the first occurrence of t_1 precedes all values of t_2 in the result of evaluating a PosRA query Q on a po-database D, it suffices to solve the `POSS` problem for the $\mathrm{PosRA}^{\mathrm{acc}}$ query $\mathrm{accum}_{h,\oplus}(Q)$ with D and the candidate value $\top$.

C Proofs for Section 5 (General Complexity Results)

C.1 Proofs of Theorems 15 and 16

▶ **Theorem 15.** *The* `POSS` *problem is NP-complete for* PosRA *and for* $\mathrm{PosRA}^{\mathrm{acc}}$*.*

▶ **Theorem 16.** `CERT` *is coNP-complete for* $\mathrm{PosRA}^{\mathrm{acc}}$ *queries.*

We first show the upper bounds:

▶ **Proposition 55.** *For any* $\mathrm{PosRA}^{\mathrm{acc}}$ *query Q,* `POSS` *for Q is in NP and* `CERT` *for Q is in co-NP.*

Proof. To show the NP membership of `POSS`, evaluate in PTIME the query without accumulation using Proposition 6, 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, check that it achieves the candidate result. Otherwise, evaluate the accumulation (in PTIME as the accumulation operator satisfies PTIME-evaluability), and check that the correct result is obtained.

To show the co-NP membership of `CERT`, follow the same reasoning but guessing an order that achieves a result different from the candidate result. ◀

We now point to the proofs of the lower bounds. For Theorem 15, the lower bound for PosRA queries follows from Theorem 22, proven in Section D.1.2; the lower bound for $\mathrm{PosRA}^{\mathrm{acc}}$ queries follows from it, by using the identity accumulation map and concatenation as accumulation (as in the proof of Theorem 17 below). For Theorem 16, the lower bound for $\mathrm{PosRA}^{\mathrm{acc}}$ queries follows from Theorem 31 for `CERT`, proven in Section E.2.2

C.2 Proof of Theorem 17

▶ **Theorem 17.** `CERT` *is in PTIME for* PosRA *queries.*

Proof. By Theorem 28 (proven in Section E.1), we know that the `CERT` problem is in PTIME for $\mathrm{PosRA}^{\mathrm{acc}}$ queries which perform accumulation in a *cancellative* monoid (see Definition 27).

To prove Theorem 17, let Q be the PosRA query of interest. Let k be the arity of its result. We will use the identity accumulation operator. Consider the monoid where $\mathcal{M}$ consists of the totally ordered relations on $\mathcal{D}^k$, that is, the finite sequences of elements of $\mathcal{D}^k$, the neutral element ε is the empty sequence, and the associative operation $\oplus$ is concatenation. This clearly defines a monoid, and it is clearly cancellative. Hence, consider the query $Q' := \mathrm{accum}_{h,\oplus}(Q)$, with $\oplus$ defined in this way, and with h being a rank-invariant accumulation map that maps each tuple to the singleton totally ordered relation containing precisely one tuple with that value. It is clear that any totally ordered relation L is a possible world of the PosRA query Q iff L is a possible result of the $\mathrm{PosRA}^{\mathrm{acc}}$ query Q'. Now, we know that `CERT` for Q' is in PTIME, because it is a $\mathrm{PosRA}^{\mathrm{acc}}$ query that performs accumulation in a cancellative monoid, so we can use Theorem 28. Hence, the `CERT` problem for Q is in PTIME as well. ◀

C.3 Proof of Theorem 18

▶ **Theorem 18.** *The position possibility and position certainty problems are in PTIME.*

Proof. Given an instance of the position possibility or certainty problem for Q, which includes a po-database D, we first compute a po-relation Γ such that $pw(\Gamma) = Q(D)$ in PTIME by Proposition 6.

Now, considering the po-relation $\Gamma = (ID, T, <)$, we can compute in PTIME, for every element $x \in ID$, its *earliest index* $i^-(x)$, which is its number of ancestors by $<$ plus one, and its *latest index* $i^+(x)$, which is the number of elements of Γ minus the number of descendants of x. It is easily seen that for any element $x \in ID$, there is a linear extension of Γ where x appears at position $i^-(x)$, or at position $i^+(x)$, or in fact at any position of $[i^-(x), i^+(x)]$, the *interval* of x.

Hence, position possibility and position certainty for tuple t and position k can be decided by checking whether some element of the order whose interval contains k has value t, or whether all such elements have value t. This concludes the proof for position possibility and certainty. ◀

D Proofs for Section 6 (Tractable Cases for POSS on PosRA)

D.1 Totally Ordered Inputs

D.1.1 Tractability Result: Proof of Theorems 19 and 21

The point of restricting to $\mathrm{PosRA}_{\mathtt{LEX}}$ queries is that they can only make the width increase in a way that depends on the *width* of the input relations, but not on their *size*:

▶ **Proposition 56.** *Let $k \geqslant 2$ and Q be a $\mathrm{PosRA}_{\mathtt{LEX}}$ query. Let $k' = k^{|Q|+1}$. For any po-database D of width $\leqslant k$, the po-relation $Q(D)$ has width $\leqslant k'$.*

Proof. We prove by induction on the $\mathrm{PosRA}_{\mathtt{LEX}}$ query Q that one can compute a bound on the width of the output of the query as a function of the bound k on the width of the inputs. For the base cases:

- Input po-relations have width $\leqslant k$.
- Constant po-relations have width 0 (for the empty po-relation) or 1 (for singletons and for constant chains).

  For the induction step:

- Given two po-relations Γ_1 and Γ_2 with bounds k_1 and k_2, their union $\Gamma_1 \cup \Gamma_2$ clearly has bound $k_1 + k_2$, as any antichain in the union must be the union of an antichain of Γ_1 and of an antichain of Γ_2.
- Given a po-relation Γ_1 with bound k_1, applying a projection or selection to Γ_1 cannot make the width increase.
- Given two po-relations Γ_1 and Γ_2 with bounds k_1 and k_2, their product $\Gamma := \Gamma_1 \times_{\mathtt{LEX}} \Gamma_2$ has bound $k_1 \cdot k_2$. To show this, consider any set A of $> k_1 \cdot k_2$ tuples in Γ, which we see as pairs of a tuple of Γ_1 and a tuple of Γ_2. It is immediate that one of the following must hold:

  1. Letting $S_1 := \{u \mid \exists v, (u, v) \in A\}$, we have $|S_1| > k_1$
  2. There exists u such that, letting $S_2(u) := \{v \mid (u, v) \in A\}$, we have $|S_2| > k_2$

  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 is a subset of tuples of Γ_1 of cardinality $> k_1$ and Γ_1 has width k_1, it cannot be an antichain, so it must contain two comparable elements $u_1 < u_2$, so that, considering v_1 and v_2 such that $a_1 = (u_1, v_1)$ and $a_2 = (u_2, v_2)$ are in A, we have by definition of $\times_{\mathtt{LEX}}$ that $a_1 <_\Gamma a_2$, so that A is not an antichain of Γ.

In the second case, as $S_2(u)$ is a subset of tuples of Γ_2 of cardinality $> k_2$ and Γ_2 has width k_2, it cannot be an antichain, so it must contain two comparable elements $v_1 < v_2$. Hence, considering $a_1 = (u, v_1)$ and $a_2 = (u, v_2)$ which are in A, we have $a_1 <_\Gamma a_2$, and again A is not an antichain of Γ.

Hence, we deduce that no set of cardinality $> k_1 \cdot k_2$ of Γ is an antichain, so that Γ has width $\leqslant k_1 \cdot k_2$, as desired.

Letting o be the number of product operators in Q plus the number of union operators, it is now clear that we can take $k' = k^{o+1}$. Indeed, po-relations with no product or union operators have width at most k (using that $k \geqslant 1$). As projections and selections do not change the width, the only operators to consider are product and union. If Q_1 has o_1 operators and Q_2 has o_2 operators, bounding by induction the width of $Q_1(D)$ to be k^{o_1+1} and $Q_2(D) = k^{o_2+1}$, for $Q = Q_1 \cup Q_2$, the number of operators is $o_1 + o_2 + 1$, and the new bound is $k^{o_1+1} + k^{o_2+1}$, which as $k \geqslant 2$ is less than $k^{o_1+1+o_2+1}$, that is, $k^{(o_1+o_2+1)+1}$. For $\times_{\mathtt{LEX}}$, we proceed in the same way and directly obtain the $k^{(o_1+o_2+1)+1}$ bound. Hence, we can indeed take $k' = k^{|Q|+1}$. ◀

From this, we will deduce POSS is tractable for $\mathrm{PosRA}_{\mathtt{LEX}}$ queries when the input po-database consists of relations of bounded width. We now prove Theorem 21, which clearly generalizes Theorem 19. We will prove both the result for $\mathrm{PosRA}_{\mathtt{LEX}}$ queries and its extension to $\mathrm{PosRA}^{\mathrm{acc}}_{\mathtt{LEX}}$ queries with finite accumulation (Theorem 32).

▶ **Theorem 21.** *Let k be a (constant) positive integer. If the input po-database is of width bounded by k, then* POSS *is in PTIME for* $\mathrm{PosRA}_{\mathtt{LEX}}$ *queries.*

Let Γ be a po-relation, such that $pw(\Gamma)$ is the result of evaluating the query Q of interest, excluding the accumulation operator, if any (so this amounts to evaluating a $\mathrm{PosRA}_{\mathtt{LEX}}$ query). We can compute this in PTIME using Proposition 6. Letting k' be the constant (only depending on Q and k) given by Proposition 56, we know that $w(\Gamma) \leqslant k'$.

We first show the tractability of POSS and CERT for $\mathrm{PosRA}^{\mathrm{acc}}_{\mathtt{LEX}}$ queries with finite accumulation, which amounts to applying directly a finite accumulation operator to Γ. We then deal with $\mathrm{PosRA}_{\mathtt{LEX}}$ queries, which amounts to solving directly POSS and CERT on the po-relation Γ.

$\mathrm{PosRA}^{\mathrm{acc}}_{\mathtt{LEX}}$ queries with finite accumulation. It suffices to show the following rephrasing of the result:

▶ **Theorem 57.** *For any constant $k' \in \mathbb{N}$, and accumulation operator* $\mathrm{accum}_{h,\oplus}$ *with finite domain, we can compute in PTIME, for any input po-relation Γ such that $w(\Gamma) \leqslant k'$, the set* $\mathrm{accum}_{h,\oplus}(\Gamma)$.

Indeed, once the possible results are determined, it is immediate to solve possibility and certainty.

For this, we need the following notions:

▶ **Definition 58.** A *chain partition* of a poset P is a partition $\mathbf{L} = (L_1, \ldots, L_n)$ of the elements of P, i.e., $P = L_1 \sqcup \cdots \sqcup L_n$, such that each L_i is a total order. (However, P may feature comparability relations not present in the L_i, i.e., relating elements in L_i to elements in L_j for $i \neq j$.) The *width* of the partition $\mathbf{L} = (L_1, \ldots, L_n)$ is n.

▶ **Definition 59.** Given a poset P, an *order ideal* of P is a subset S of P such that, for all $x, y \in P$, if $x < y$ and $y \in S$ then $x \in S$.

We also need the following known results:

▶ **Theorem 60 [Dil50].** *Any poset P has a chain decomposition of width $w(P)$.*

▶ **Theorem 61 [Ful55].** *For any poset P, we can compute in PTIME a chain decomposition of P of minimal width.*

We now prove Theorem 57:

Proof of Theorem 57. Consider a po-relation $\Gamma = (ID, T, <)$, with underlying poset $P = (ID, <)$. Using Theorems 60 and 61, compute in PTIME a chain decomposition $\mathbf{L}$ of P of width k'. For $1 \leqslant i \leqslant k'$, write $n_i := |L_i|$, and for $0 \leqslant j \leqslant n_i$, write $L_i^{\leqslant j}$ to denote the subset of L_i containing the first j elements of the chain (in particular $L_i^{\leqslant 0} = \emptyset$).

We now consider all vectors of the form $(m_1, \ldots, m_{k'})$, with $0 \leqslant m_i \leqslant n_i$, of which there are polynomially many (there are $\leqslant |\Gamma|^{k'}$, where k' is constant). To each such vector $\mathbf{m}$ we associate the subset $s(\mathbf{m})$ of P consisting of $\bigsqcup_{i=1}^{k'} L_i^{\leqslant m_i}$.

We call such a vector $\mathbf{m}$ *sane* if $s(\mathbf{m})$ is an order ideal. (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 overall because of the additional comparability relations across the chains that may be featured in P.) For each vector $\mathbf{m}$, we can check in PTIME whether it is sane, by materializing $s(\mathbf{m})$ and checking that it is an ideal for each comparability relation (of which there are $O(|P|^2)$).

By definition, for each sane vector $\mathbf{m}$, $s(\mathbf{m})$ is an ideal. We now observe that the converse is true, and that for every ideal S of P, there is a sane vector $\mathbf{m}$ such that $s(\mathbf{m}) = S$. To see why, consider an ideal S, and determine for each chain L_i the last element of the chain present in the ideal; let m_i be its position in the chain. S then does not include any element of L_i at a later position, and because L_i is a chain it must include all elements before, hence, $S \cap L_i = L_i^{\leqslant m_i}$. As $\mathbf{L}$ is a chain decomposition of P, this entirely determines S. Thus we have indeed $S = s(\mathbf{m})$, and the fact that $s(\mathbf{m})$ is sane is witnessed by S.

For any sane vector $\mathbf{m}$, we now write $t(\mathbf{m}) := \mathrm{accum}_{h,\oplus}(T(s(\mathbf{m})))$ (recall that T maps elements of the poset to tuples, and can therefore naturally be extended to map sub-posets to sub-po-relations). This is a subset of the accumulation domain $\mathcal{M}$ (since the latter is finite, this subset is of constant size). It is immediate that $t((0, \ldots, 0)) = \varepsilon$, the neutral element of the accumulation monoid, and that $t((n_1, \ldots, n_{k'})) = \mathrm{accum}_{h,\oplus}(\Gamma)$ is our desired answer. Denoting by e_i the vector consisting of $n-1$ zeroes and a 1 at position i, for $1 \leqslant i \leqslant k'$, we now observe that, for any sane vector $\mathbf{m}$, we have:

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

where the operator "$-$" is the component-by-component tuple difference and where we define $t(\mathbf{m} - e_i)$ to be $\emptyset$ if $\mathbf{m} - e_i$ is not sane or if one of the coordinates of $\mathbf{m} - e_i$ is < 0. Equation 1 holds because 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 $L_i[m_i]$ for $1 \leqslant i \leqslant m$ such that $m_i \geqslant 1$, 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, otherwise the removed $L_i[m_i]$ was not actually maximal because it was comparable to (and smaller than) some $L_j[m_j]$ for $j \neq i$). 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.

This last equation allows us to compute $t(n_1, \ldots, n_{k'})$ in PTIME by a dynamic algorithm, enumerating 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. Hence, we have computed $\mathrm{accum}_{h,\oplus}(\Gamma)$ in PTIME, which concludes the proof. ◀

PosRA$_{\texttt{LEX}}$ queries. First note that, for queries with no accumulation, we cannot reduce `POSS` and `CERT` to the case with accumulation, because the monoid of tuples under concatenation does not satisfy the hypothesis of finite accumulation. Hence, we need specific arguments to prove Theorem 21 for queries with no accumulation.

Recall that the `CERT` problem is in PTIME for such queries by Theorem 17, so it suffices to study the case of `POSS`. We do so by the following result, which is obtained by adapting the proof of Theorem 57:

▶ **Theorem 62.** *For any constant $k \in \mathbb{N}$, we can determine in PTIME, for any input po-relation Γ such that $w(\Gamma) \leqslant k$ and totally ordered relation L, whether $L \in pw(\Gamma)$.*

Proof. The proof of Theorem 57 adapts because of the following: to decide instance possibility, we do not need to compute *all* possible accumulation results (which may be exponentially numerous), but it suffices to store, for each sane vector $\mathbf{m}$, whether the prefix of the correct length of the candidate possible world can be achieved in the order ideal $s(\mathbf{m})$. More formally, we define $t((0,\ldots,0)) :=$ true, and:

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

where L is the candidate possible world. We conclude by a dynamic algorithm as in Theorem 57. ◀

This concludes the proof of Theorem 21, and, as an immediate corollary, of Theorem 19.

D.1.2 Hardness result: Proof of Theorem 22

▶ **Theorem 22.** *The `POSS` problem is NP-complete for PosRA$_{\texttt{DIR}}$ queries, even when the input po-database is restricted to consist only of totally ordered po-relations.*

Proof. The reduction is from the UNARY-3-PARTITION problem [GJ79]: 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.

Fix $\mathcal{D} := \mathbb{N} \sqcup \{\mathsf{s}, \mathsf{n}, \mathsf{e}\}$, standing for *start*, *inner*, and *end*. Let S be the totally ordered po-relation $\mathbb{N}^*_{\leqslant 3m-1}$, and let S' be the totally ordered po-relation constructed from the instance $\mathcal{I}$ as follows: for $1 \leqslant i \leqslant 3m$, we consider the concatenation of one tuple t^i_1 with value s, n_i tuples t^i_j (with $2 \leqslant j \leqslant n_i + 1$) with value n, and one tuple $t^i_{n_i+2}$ with value e, and S' is the total order formed 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 coming from relation S'. Note that S' is an *input* relation, not the constant expression that gives the same relation.

We define the candidate possible world as follows:

- L_1 is a totally ordered 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 totally ordered relation defined as above, except that $3m - i$ is replaced by $i - 1$.
- L' is the totally ordered relation defined as the concatenation of m copies of the following sublist: three tuples with value s, B tuples with value n, three tuples with value e.
- L is the concatenation of L_1, L', and L_2.

We now consider the POSS instance that asks whether L is a possible world of the query $Q(S, S')$, where S and S' are the input totally ordered relations. We claim that this instance is positive iff the original UNARY-3-PARTITION instance $\mathcal{I}$ is positive. As the reduction process described above is clearly PTIME, this suffices to show our desired hardness result, so all that remains to show our hardness result for $\mathrm{PosRA}_{\mathtt{DIR}}$ is to prove this claim. We now do so.

Denote by R the po-relation obtained by evaluating $Q(S, S')$, and note that all tuples of R 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 R *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 R that occur in the first k positions of L'', and say that the other elements are *unmatched*. We call the *i-th row* of R the elements whose first component before projection was $i - 1$: note that, for each i, R imposes a total order on the i-th row.

We first observe that for any linear extension L'' realizing $L_1^{\leqslant k}$, for all i, writing the i-th row as $t'_1 < \ldots < t'_{|S'|}$, the unmatched elements must be all of the form t'_j for $j > k_i$ for some k_i, 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 R would have been violated when enumerating L''. Further, by cardinality we clearly have $\sum_i k_i = k$.

Second, when a linear extension L'' of R realizes $L_1^{\leqslant k}$, we say that we are in a *whole situation* if for all i, the value of element t'_{k_i+1} is either undefined (i.e., there are no row-i unmatched elements, which means $k_i = |S'|$) or it is s. This clearly implies that k_i is of the form $\sum_{j=1}^{l_i}(n_j + 2)$ for some l_i; we call $S_i := \biguplus_{1 \leqslant j \leqslant l_i} \{\{\, n_j \,\}\}$ the bag of *row-i consumed integers*. The *row-i remaining integers* are $E \backslash S_i$ (seeing E as a multiset).

We now prove the following claim: for any linear extension of R realizing L_1, we are in a whole situation, and the multiset union $\biguplus_{1 \leqslant i \leqslant 3m} S_i$ is equal to the multiset obtained by repeating integer n_i of E $3m - i$ times 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 $l' > l$, you 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 you are in a whole situation 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, and it is easily seen that this must bring you 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 this particular row can be used, as any first unmatched element (if any) of another row is s. Hence the claim is proven.

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 R whose first $|L_1|$ tuples achieve L_1 and whose last $|L_2|$ tuples achieve L_2, the row-i unmatched elements are a contiguous sequence t'_j with $k_i < j < m_i$ for some k_i and m_i. In addition, if we have $k_i < m_i - 1$, then t'_{k_i} has value e and t'_{m_i} has value s, and the unmatched values (defined in an analogous fashion) are a multiset corresponding exactly to $\{\{\, n_1, \ldots, n_{3m} \,\}\}$. So the unmatched elements when having read L_1 (at the beginning) and L_2 (at the end) are formed of $3m$ totally ordered sequences, of length $n_i + 2$ for $1 \leqslant i \leqslant 3m$, of the form s, n_i times n, and e, with a certain order relation between the elements of the sequences (arising from the fact that some may be on the same row, or that some may be on different rows but comparable by definition of $\times_{\mathtt{DIR}}$).

But we now notice that we can clearly 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 $3m$ totally ordered sequences (one for each row, with that of row i being $\mathtt{s}$, n_i times $\mathtt{n}$ and $\mathtt{e}$, for all i) and there are no order relations across sequences. Let T be the sub-po-relation of R that consists of exactly these unmatched elements. We denote the elements of T as u_l^j with $1 \leqslant j \leqslant 3m$ iterating over the totally ordered sequences, and $1 \leqslant l \leqslant n_j + 2$ iterating within each sequence. T is the parallel composition of $3m$ total orders, namely, $u_1^j < u_2^j < \cdots < u_{n_j+2}^j$ for all j, having values $\mathtt{s}$ for u_1^j, $\mathtt{e}$ for $u_{n_j+2}^j$, and $\mathtt{n}$ for the others.

We now claim that for any sequence L'', the concatenation $L_1 L'' L_2$ is a possible world of R if and only if L'' is a possible world of T. The "only if" direction was proved with the construction above. The "if" direction comes from the fact that T is the *least constrained* possible po-relation for the unmatched sequences, since the order on the sequences of remaining elements when matching L_1 and L_2 is known to be total. 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 T. (In other words, the point of the construction so far was to reduce $\mathtt{POSS}$ under our restrictive assumptions to $\mathtt{POSS}$ for instances of a slightly less restricted kind, namely, the parallel composition of an unbounded number of total orders of unbounded length.)

To see why this last claim holds, observe that there is a bijection between 3-partitions of E and linear extensions of T which achieve L'. Indeed, consider a 3-partition $\mathbf{s} = (s_1^i, s_2^i, s_3^i)$ for $1 \leqslant i \leqslant m$, with $n_{s_1^i} + n_{s_2^i} + n_{s_3^i} = B$ for all i, and each element of E occurring exactly once in $\mathbf{s}$. We can realize L' from $\mathbf{s}$, picking successively the following for $1 \leqslant i \leqslant m$: the tuples $u_1^{s_p^i}$ for $1 \leqslant p \leqslant 3$ that have value $\mathtt{s}$; the tuples $u_j^{s_p^i}$ for $1 \leqslant p \leqslant 3$ and $2 \leqslant j \leqslant n_{s_p^i} + 1$ that have value $\mathtt{n}$ (hence, B tuples in total); the tuples $t^{s_p^i}_{n_{s_p^i}+2}$ for $1 \leqslant p \leqslant 3$ that have value $\mathtt{e}$. Conversely, it is easy to build a 3-partition from any linear extension to achieve L' from T. This proves our last claim, and concludes the proof. ◀

D.2 Unordered Inputs

D.2.1 Auxiliary Result on Ia-Width: Proof of Proposition 25

We first show a preliminary result about indistinguishable sets:

▶ **Lemma 63.** *For any poset $(V, <)$ and indistinguishable sets $S_1, S_2 \subseteq V$ such that $S_1 \cap S_2 \neq \emptyset$, $S_1 \cup S_2$ is an indistinguishable set.*

Proof. Let $x, y \in S_1 \cup S_2$ and $z \in V \backslash (S_1 \cup S_2)$, assume that $x < z$ and show that $y < z$. As S_1 and S_2 are indistinguishable sets, this is immediate unless $x \in S_1 \backslash S_2$ and $y \in S_2 \backslash S_1$, or vice-versa. We assume the first case as the second one is symmetric. Consider $w \in S_1 \cap S_2$. As $x < z$, we know that $w < z$ as S_1 is an indistinguishable set, so that $y < z$ as S_2 is an indistinguishable set, which proves the desired implication. The fact that $z < x$ implies $z < y$ is proved in a similar fashion. ◀

The lemma implies:

▶ **Corollary 64.** *For any poset $(V, <)$ and indistinguishable antichains $A_1, A_2 \subseteq V$ such that $A_1 \cap A_2 \neq \emptyset$, $A_1 \cup A_2$ is an indistinguishable antichain.*

Proof. We only need to 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 contradicts the fact that A_2 is an antichain. ◀

We also show:

▶ **Lemma 65.** *For any poset $(V, <)$ and indistinguishable antichain A, for any $A' \subseteq A$, 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 $z \in V \backslash A'$, and show that $x < z$ implies $y < z$ (the other three implications are symmetric). If $z \in V \backslash A$, we conclude because A is an indistinguishable set. If $z \in A \backslash A'$, we conclude because, as A is an antichain, z is incomparable both to x and to y. ◀

We can now state and prove the Proposition:

▶ **Proposition 25.** *The ia-width of any poset, and a corresponding ia-partition, can be computed in PTIME.*

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, and check in PTIME if it is an antichain, and if it is an indistinguishable set). Repeat the process while it is possible to merge classes (i.e., at most linearly many times). This process concludes in PTIME.

Now assume that there is a partition of strictly smaller cardinality. There has to be a class c of this partition which intersects two different classes $c_1 \neq c_2$ of the original partition, otherwise it is a refinement of the previous partition and so has a higher number of classes. But now, by Corollary 64, $c \cup c_1$ and $c \cup c_2$ are indistinguishable antichains, and thus $c \cup c_1 \cup c_2$ also is. Now, by Lemma 65, this implies that $c_1 \cup c_2$ is an indistinguishable antichain. Now, when constructing our original ia-partition, the algorithm has considered one element of c_1 and one element of c_2, attempted to merge the classes, and, since it has not merged them, $c_1 \cup c_2$ was not an indistinguishable antichain; yet, we have just proved that it was, a contradiction. ◀

D.2.2 Tractability Result: Proof of Theorems 23 and 26

As already mentioned, Theorem 23 is a direct corollary of the more general result:

▶ **Theorem 26.** *For any $k \in \mathbb{N}$, `POSS` is in PTIME for* PosRA *queries assuming that input po-databases have ia-width $\leqslant k$.*

We now prove Theorem 26. Once again, as in the proof of Theorem 21 (Appendix D.1.1), we use Proposition 6 to evaluate in PTIME the accumulation-free part of the query Q to a po-relation Γ. We will show that the result of this query has bounded ia-width, with the following general result:

▶ **Proposition 66.** *For any* PosRA *query Q and $k \in \mathbb{N}$, there is $k' \in \mathbb{N}$ such that for any po-database D of ia-width $\leqslant k$, the po-relation $Q(D)$ has ia-width $\leqslant k'$.*

Proof. We compute the bound k' by induction. For the base cases:

- The input relations have ia-width at most k.
- The constant relations have constant ia-width with the trivial ia-partition.

For the induction step:

- Projection clearly does not change ia-width.

- Selection may only reduce the ia-width: the image of an ia-partition of the original relation is an ia-partition of the selection, and it cannot have more classes.
- The union of two relations with ia-width c_1 and c_2 has ia-width at most $c_1 + c_2$, taking an ia-partition of the union as the union of ia-partitions of the operands.
- The $\times_{\texttt{DIR}}$ or $\times_{\texttt{LEX}}$ product of two relations Γ_1 and Γ_2 with ia-width respectively $\leqslant c_1$ and $\leqslant c_2$ is $\leqslant c_1 \cdot c_2$. Indeed, create partition the result of the product by creating one class per pair of classes of each input relation. Now, observe that it is clear that if $\langle t_1, t_2 \rangle$ and $\langle t'_1, t'_2 \rangle$ are in the same class of the product, then they are incomparable, because t_1 and t'_1, and t_2 and t'_2, are in the same class of the ia-partitions of Γ_1 and Γ_2 respectively, hence incomparable. Further, it is clear that the order relation between any $\langle t_1, t_2 \rangle$ and $\langle t'_1, t'_2 \rangle$ in the product only depends on the order relation between t_1 and t'_1, t_2 and t'_2, which only depends by indistinguishability on the classes of t_1 and t'_1, and t_2 and t'_2, in the ia-partitions of Γ_1 and Γ_2 respectively. This shows that the partition of the product that we have defined is indeed an ia-partition of the product, and it has size $\leqslant c_1 \cdot c_2$.

Further, we show as for Proposition 56 that the bound is $\max(q, k, 2)^{o+1}$ where o is the number of unions and products of the query, and q is the largest value such that $\mathbb{N}^*_{\leqslant q}$ appears in the query Q (taking $q = 0$ if no $\mathbb{N}^*_{\leqslant \bullet}$ appears in Q). Indeed, input relations have ia-width at most k, and constant relations have ia-width at most $q \leqslant k$, so, if we take $\max(k, q, 2)$ as a global bound, the worst composition operations are products, which yields the desired bound. ◀

Now that we know that the resulting relation Γ has ia-width bounded by a constant $k \in \mathbb{N}$, we will again study first the case of finite and rank-invariant PosRA$^{\text{acc}}$ queries (with aggregation directly applied to the po-relation Γ), and then PosRA queries, where it suffices to study `POSS` (and solve it directly on Γ).

PosRA$^{\text{acc}}$ queries with finite and rank-invariant accumulation. It suffices to show the following rephrasing of the result:

▶ **Theorem 67.** *For any constant $k \in \mathbb{N}$, and finite and rank-invariant accumulation operator $\mathrm{accum}_{h,\oplus}$, we can compute in PTIME, for any input po-relation Γ with ia-width $\leqslant k$, the set $\mathrm{accum}_{h,\oplus}(\Gamma)$.*

Proof. We consider the constant-size partial order P' on the classes of the ia-partition of the underlying poset of Γ. For each class, we consider a constant-size vector indicating, for each possible $\alpha \in \mathcal{M}$, the number of elements v of Γ such that $h(v, \cdot) = \alpha$ which have already been enumerated in the class (thanks to rank-invariance, we know that h does not depend on its second argument). Clearly the number of such vectors is polynomial, and they uniquely describe all possible ideals of the relation, up to identifying ideals that only differ by elements in the same class which are mapped to the same value by h (They also describe some subsets which are not ideals.)

We use a dynamic programming approach in the same way as in the proof of Theorem 57. Indeed, we can enumerate the polynomial number of vectors and compute for each of them in PTIME whether it actually describes an ideal, and we can determine exactly the possible accumulation results for each vector as a function of those of the preceding vectors in the lexicographic order. We use the PTIME-evaluability of the underlying accumulation monoid to ensure that all computations of accumulation results can be performed in PTIME, again by storing with each accumulation result a witnessing totally ordered relation from which the result is computed in PTIME, which is a prefix of a linear extension of Γ. ◀

PosRA queries. Now, for PosRA queries, once again the `CERT` problem is tractable by Theorem 17. For `POSS`, we prove the following, using an entirely different approach (again because we cannot use the identity monoid as it is not finite):

▶ **Proposition 68.** *For any constant $k \in \mathbb{N}$, we can determine in PTIME, for any input po-relation Γ with ia-width $\leqslant k$ and totally ordered po-relation L, whether $L \in pw(\Gamma)$.*

Proof. Let $P = (c_1, \ldots, c_k)$ be an ia-partition of width k of Γ, which can be computed in PTIME by Proposition 25.

If there is a way to realize L as a possible world of Γ, we call the *finishing order* the permutation π of $\{1, \ldots, k\}$ obtained by considering, for each class c_i of P, the largest position n_i of $\{1, \ldots, |L|\}$ to which an element of c_i is mapped, and sorting the class indexes by ascending finishing order. We say we can realize L with finishing order π if there is a realization of L whose finishing order is π. Hence, it suffices to check, for every possible permutation π, whether L can be realized from Γ with finishing order π: this does not make the complexity worse as 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 which have already been used (distinguish the *used* and *unused* elements; initially all elements are unused), and distinguish the classes of P in *exhausted classes*, the ones where all elements have been mapped; *open classes*, the ones where all smaller elements have been mapped; and *blocked classes*, the ones where some smaller element is not mapped (initially the open classes are those which are roots in the poset obtained from the underlying poset of Γ by quotienting by the equivalence relation induced by P; and the others are blocked).

When reading a value v from L, consider all open classes. If none of these classes have an unused element with value v, reject, i.e., conclude that we cannot realize L as a possible world of Γ with finishing order π. Otherwise, take the open class with the lowest finishing time (i.e., appears the earliest in π) that has such an element, and use an arbitrary suitable element from it. (Update the class to be *exhausted* if it is, in which case update from *blocked* to *open* the classes that must be). Once L is read, accept iff all elements are used (i.e., all classes are exhausted).

It is clear by construction that if this greedy algorithm accepts then it has found a way to match L in Γ; indeed all matches that it performs satisfy the values and the order relations of Γ. It must now be proved that if L can be matched in Γ with finishing order π, then the algorithm accepts when considering π. To do so, we must show that if there is such a match, then there is such a match where all elements are mapped, following what the greedy algorithm does, to a suitable element in the open class with smallest finishing time (we call this a *minimal* element); if we can prove this, then this justifies the existence of a match that the greedy algorithm will construct (we call this a *greedy match*).

Now, to see why this is possible, consider a match m and take the smallest element t of L mapped to an element s in class c in Γ which is non-minimal, i.e., these is a minimal element s' in class $c' \neq c$ that has the same value, and $\pi(c') < \pi(c)$, i.e., c' finishes earlier than c according to π. Let t' be the element to which s' is mapped by m (and $t <_L t'$). Consider the match m' obtained by mapping t to s' and t' to s. The new match m' still satisfies conditions on the values because s and s' have the same value. If we can show that m' additionally satisfies the order constraints of Γ, then we will have justified the existence of a match that differs from a greedy match at a later point; so, reapplying the rewriting argument, we will deduce the existence of a greedy match. So it only remains to show that m' satisfies the order constraints of Γ.

Let us assume by way of contradiction that m' violates an order constraint of Γ. The only possible kind of violation, given that m had no violation, is that some t'' of L, $t <_L t'' <_L t'$, is matched to s'' in Γ for which we have $s < s''$ (so this order constraint of Γ is respected by m but not by m'). Now, using indistinguishability of elements in c, if s'' was thus mapped in m, it means that the class c of s was exhausted when reaching t'' in L: indeed, as $s < s''$, any non-matched element of c would be an ancestor of s'' and prevent us from mapping t''

to it. Now, because t' was not reached yet in m, the class c' of s' was not exhausted yet. However, this contradicts the fact that c' finishes before c according to π. So m' also satisfies the order constraints.

This shows that we can rewrite m to a greedy match, which the greedy algorithm will find. This concludes the proof. ◀

E Proofs for Section 7 (Tractable Cases for $\mathrm{PosRA}^{\mathrm{acc}}$)

E.1 Cancellative Monoids

▶ **Theorem 28.** `CERT` *is in PTIME for* $\mathrm{PosRA}^{\mathrm{acc}}$ *with accumulation in a cancellative monoid.*

We formalize the definition of possible ranks for pairs of incomparable elements, and of the *safe swaps* property:

▶ **Definition 69.** Given two *incomparable* elements x and y in Γ, their *possible ranks* $\mathrm{pr}_\Gamma(x, y)$ is the interval $[a + 1, |\Gamma| - d]$, where a is the number of elements that are either ancestors of x or of y in Γ (not including x and y), and d is the number of elements that are either descendants of x or of y (again excluding x and y themselves).

Let $(\mathcal{M}, \oplus, \varepsilon)$ be an accumulation monoid and let $h : \mathcal{D} \times \mathbb{N} \to \mathcal{M}$ be an accumulation map. The po-relation Γ has the *safe swaps* property with respect to $\mathcal{M}$ and h if the following holds: for any pair $t_1 \neq t_2$ of incomparable tuples of Γ, for any pair $p, p + 1$ of *consecutive* integers in $\mathrm{pr}_\Gamma(t_1, t_2)$, we have:

$$h(t_1, p) \oplus h(t_2, p + 1) = h(t_2, p) \oplus h(t_1, p + 1)$$

We first show the following soundness result for possible ranks:

▶ **Lemma 70.** *For any poset P and incomparable elements $x, y \in P$, for any $p \neq q \in \mathrm{pr}_P(x, y)$, there exists a linear extension L of P such that element x is enumerated at position p in L, and element y is enumerated at position q, and we can compute it in PTIME from P.*

Proof. We can construct the desired linear extension L by starting to enumerate all elements which are ancestors of either x or y in any order, and finishing by enumerating all elements which are descendants of either x or y, in any order: that this can be done without enumerating either x or y follows from the fact that x and y are incomparable.

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

All unenumerated elements are either x, y, or incomparable to both x and y. Consider any linear extension of the unenumerated elements except x and y; it has length $|P| - d - a - 2$. Now, as $p' \neq q'$, if $p' < q'$, 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 the linear extension. We proceed similarly, reversing the roles of x and y, if $q' < p'$. The overall process is clearly in PTIME. ◀

We can then show:

▶ **Lemma 71.** *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 (t_1, t_2) of elements of Γ, of which there are quadratically many. Check in PTIME whether they are incomparable. If yes, compute in PTIME $\mathrm{pr}_\Gamma(t_1, t_2)$, and consider each pair p, $p + 1$ of consecutive integers (there are linearly many).

Using Lemma 70, construct in PTIME a possible world L of Γ where t_1 and t_2 occur respectively at positions p and $p + 1$. By definition, using associativity of the composition law, the result of accumulation on L is $w := v \oplus h(t_1, p) \oplus h(t_2, p+1) \oplus v'$, where v is the result of accumulation on the tuples in L before t_1, and v' is the result of accumulation on the tuples in L after t_2. As the accumulation operator satisfies PTIME-evaluability, we can compute w in PTIME from L.

Now, by symmetry of the definition of pr_Γ, it is clear that we have $p, p+1 \in \mathrm{pr}_\Gamma(t_2, t_1)$, so using Lemma 70 again we obtain in PTIME a possible world L' where t_2 and t_1 occur respectively at positions p and $p+1$; further, from the proof of Lemma 70 it is clear that L' can be constructed to be equal to L except at positions p and $p+1$. Hence, the result of accumulation on L' is $w' := v \oplus h(t_2, p) \oplus h(t_1, p+1) \oplus v'$, which we again compute in PTIME thanks to PTIME-evaluability.

Now, as $\mathcal{M}$ is cancellative, v is cancellable, so, for any $a, b \in \mathcal{M}$, if $v \oplus a = v \oplus b$ then $a = b$; conversely, it is obvious that if $a = b$ then $v \oplus a = v \oplus b$. Likewise, by cancellativity of v', we have $v \oplus a \oplus v' = v \oplus b \oplus v'$ iff $a = b$, for any $a, b \in \mathcal{M}$. This means that we can test whether t_1, t_2, p and $p+1$ are a violation of the safe swaps criterion by checking whether $w \neq w'$. ◀

Now it is easily seen that Theorem 28 is implied by the following claim.

▶ **Proposition 72.** *If the monoid $(\mathcal{M}, \oplus, \varepsilon)$ is cancellative, then, for any po-relation Γ, we have $|\mathrm{accum}_{h,\oplus}(\Gamma)| = 1$ iff Γ has safe swaps with respect to $\oplus$ and h.*

Indeed, given an instance (D, v) of the `CERT` problem for query Q, we can find Γ such that $pw(\Gamma) = Q(D)$ in PTIME by Proposition 6, and we can test in PTIME by Lemma 71 whether Γ has safe swaps with respect to $\oplus$ and h. If it does not, then, by the above claim, we know that v cannot be certain, so (D, v) is not a positive instance of `CERT`. If it does, then, by the above claim, $Q(D)$ has only one possible result, so to determine whether v is certain it suffices to compute any linear extension of Γ, obtaining one possible world L of $Q(D)$, and checking whether accumulation on L yields v. If it does not, then (D, v) is not a positive instance of `CERT`. If it does, then as this is the only possible result, (D, v) is a positive instance of `CERT`.

We now prove this claim:

Proof of Proposition 72. For one direction, assume that Γ does *not* have the safe swaps property. Hence, there exist two incomparable elements t_1 and t_2 in Γ and a pair of consecutive integers $p, p+1$ in $\mathrm{pr}_\Gamma(t_1, t_2)$ such that the following disequality holds:

$$h(t_1, p) \oplus h(t_2, p+1) \neq h(t_2, p) \oplus h(t_1, p+1) \tag{2}$$

By the same reasoning as in the proof of Lemma 71, we compute two possible worlds L and L' of Γ that are identical except that t_1 and t_2 occur respectively at positions p and $p+1$ in L, and at positions $p+1$ and p respectively in L'. We then use cancellativity (as in the same proof) to deduce that L and L' are possible worlds of Γ that yield different accumulation results $w \neq w'$, so we conclude that $|\mathrm{accum}_{h,\oplus}(\Gamma)| > 1$.

For the converse direction, assume that Γ has the safe swaps property. Assume by way of contradiction that there are two possible worlds L_1 and L_2 of Γ such that the result of accumulation on L_1 and on L_2, respectively w_1 and w_2, are different, i.e., $w_1 \neq w_2$. Take L_1 and L_2 to have the longest possible common prefix, i.e., the first position i such that tuple i of L_1 and tuple i of L_2 are different is as large as possible. Let i_0 be the length of the common prefix. Let Γ' be Γ but removing the elements 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 t_1 and t_2, $t_1 \neq t_2$, be the first elements respectively of L'_1 and L'_2; it is immediate that t_1 and t_2 are roots of Γ', that is, no element of Γ' is less than them. Further, it is clear that accumulation over L'_2 (but offsetting all ranks by i_0) and accumulation over L'_1 (also offsetting all ranks by i_0), respectively w'_1 and w'_2, are different, because, by the contrapositive of cancellativity, combining them 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 of Γ' whose first element is t_1 but such that the result of accumulation on L'_3 is w'_2. If we can build such an 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 L'_3.

As t_1 is a root of Γ', L'_2 must enumerate t_1, and all elements before t_1 in L'_2 must be incomparable to t_1. Write these elements as $L''_2 = s_1, \ldots, s_m$, and write L'''_2 the sequence following t_1, so that L'_2 is the concatenation of L''_2, $[t_1]$, and L'''_2. We now consider the following sequence of totally ordered relations, which are clearly possible worlds of Γ':

- $s_1 \ldots s_m t_1 L'''_2$
- $s_1 \ldots s_{m-1} t_1 s_m L'''_2$
- $s_1 \ldots s_{m-2} t_1 s_{m-1} s_m L'''_2$
- $s_1 \ldots s_{m-3} t_1 s_{m-2} \ldots s_m L'''_2$
- $\vdots$
- $s_1 \ldots s_3 t_1 s_4 \ldots s_m L'''_2$
- $s_1 s_2 t_1 s_3 \ldots s_m L'''_2$
- $s_1 t_1 s_2 \ldots s_m L'''_2$
- $t_1 s_1 \ldots s_m L'''_2$

We can see that any consecutive pair in this list achieves the same accumulation result. Indeed, it suffices to show that the accumulation result for the only two contiguous indices where they differ is the same, and this is exactly what the safe swaps property for t_1 and s_j says, as it is easily checked that $j, j+1 \in \mathrm{pr}_{\Gamma'}(s_j, t_1)$, so that $j + i_0, j + i_0 + 1 \in \mathrm{pr}_{\Gamma}(s_j, t_1)$. Now, the first totally ordered relation in the list is L'_2, and the last totally ordered relation in this list is our desired L'_3. This concludes the second direction of the proof.

Hence, the desired equivalence is shown. ◀

This finishes the proof of Proposition 72, which, as we argued, concludes the proof of Theorem 28.

E.2 Other Restrictions on Accumulation

▶ **Theorem 31.** `POSS` *and* `CERT` *are respectively NP-hard and coNP-hard for* $\mathrm{PosRA}^{\mathrm{acc}}$ *queries performing finite and rank-invariant accumulation, even assuming that the input po-database contains only totally ordered po-relations.*

E.2.1 Proof of Theorem 31 for `POSS`

We show the following strengthening of the result, which will be useful to prove the result for `CERT` in Section E.2.2.

▶ **Proposition 73.** *There is a* $\mathrm{PosRA}^{\mathrm{acc}}$ *query* Q *with finite and rank-invariant accumulation such that the* `POSS` *problem is NP-hard for* Q*, even assuming that all input po-relations are totally ordered. Further, for any input po-database* D *(no matter whether the relations are totally ordered or not), we have* $|Q(D)| \leqslant 2$.

Define the following finite domains:

- $\mathcal{D}_- := \{\mathsf{s}_-, \mathsf{n}_-, \mathsf{e}_-\}$;
- $\mathcal{D}_+ := \{\mathsf{s}_+, \mathsf{n}_+, \mathsf{e}_+\}$;
- $\mathcal{D}_\pm := \mathcal{D}_- \sqcup \mathcal{D}_+ \sqcup \{\mathsf{l}, \mathsf{r}\}$ (the additional elements stand for “left” and “right”).

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 for any PosRA query:

▶ **Definition 74.** 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., can be seen as a word over $\mathcal{D}_{\pm}$ that satisfies e).

Note that the balanced checking problem only makes sense (i.e., is not vacuously false) for unary queries (i.e., queries whose output arity is 1) whose output tuples have value in $\mathcal{D}_{\pm}$.

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 show the following lemma:

▶ **Lemma 75.** *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, we construct the query $Q'_{\mathsf{b}}(R,S) := [\mathsf{l}] \cup_{\mathtt{CAT}} ((R \cup S) \cup_{\mathtt{CAT}} [\mathsf{r}])$, i.e., $Q'_{\mathsf{b}}(R,S)$ is the parallel composition of R and S, preceded by l and followed by r. Recall the definition of $\cup_{\mathtt{CAT}}$ (Definition 50), and recall from Lemma 51 that $\cup_{\mathtt{CAT}}$ can be expressed by a PosRA query.

We write L_w^- for any word $w \in \mathcal{D}_+^*$ to be the totally ordered unary po-relation whose only possible world is the sequence obtained by mapping each letter of w to the corresponding letter in $\mathcal{D}_-$. We claim the following:

▶ **Lemma 76.** *For any $w \in \mathcal{D}_+^*$ and unary po-relation S over $\mathcal{D}_+$, we have $w \in pw(S)$ iff $\{R \mapsto L_w^-, S \mapsto S\}$ is a positive instance to the balanced checking problem for Q'_{b}; in other words, iff $Q'_{\mathsf{b}}(L_w^-, S)$ has some balanced possible world.*

Proof. For the first direction, assume that w is indeed a possible world L of S and let us construct a balanced possible world L' of $Q'_{\mathsf{b}}(L_w^-, S)$. L' starts with l. Then, L' successively contains alternatively one tuple from L_w^- and one from L, in their total order. Finally, L' ends with r. L' is clearly balanced.

For the converse direction, observe that a balanced possible world of $Q'_{\mathsf{b}}(L_w^-, S)$ must consist of first l, last r, and, between the two, tuples alternatively enumerated from L_w^- from one of the possible worlds of S, with that possible world of S achieving w. ◀

We now use Lemma 76 to prove Lemma 75:

Proof of Lemma 75. By Theorem 22 and its proof, there is a unary query Q_0 in PosRA such that the `POSS` problem for Q_0 is NP-hard, even for input relations over $\mathcal{D}_+$ (this is by observing that the proof uses $\{\mathsf{s}, \mathsf{n}, \mathsf{e}\}$ and renaming the alphabet), and even assuming that D contains only totally ordered relations. Consider the query $Q_{\mathsf{b}}(R, D) := Q'_{\mathsf{b}}(R, Q_0(D))$; Q_{b} is a PosRA query, and by definition of Q'_{b} it satisfies the additional condition of all possible worlds satisfying 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, w) is a positive instance to `POSS` for Q_0 iff D', obtained by adding to D the relation name R that maps to the totally ordered L_w^-, is a positive instance of the balanced checking problem for Q_{b}. This is exactly what Lemma 76 shows. 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. ◀

Hence, all that remains to show is to prove Proposition 73 using Lemma 75. The idea is that we will reduce the balanced checking problem to POSS, using an accumulation operator to do the job, which will allow us to ensure that there are at most two possible results. To do this, we need to introduce some new concepts.

Let A be the deterministic complete finite automaton defined as follows, which clearly recognizes the language of the regular expression e, and let Q be its state space:

- there is a l-transition from the initial state q_i to a state q_0;
- there is a r-transition from q_0 to the final state q_f;
- for $\alpha \in \{\mathsf{s}, \mathsf{n}, \mathsf{e}\}$:
  - there is an α_+-transition from q_0 to a state q_α;
  - there is an α_--transition from q_α to q_0;
- all other transitions go to a sink state q_s.

We now define the *transition monoid* of this automaton, which is a finite monoid (so we are indeed performing finite accumulation). Let $\mathcal{F}_Q$ be the finite set of total functions from Q to Q, and consider the monoid defined on $\mathcal{F}_Q$ 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 $\mathcal{F}_Q$ as follows, which can be understood as a homomorphism from the free monoid on $\mathcal{D}_\pm^*$ 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 $q \in Q$ 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 $w \in \mathcal{D}_\pm^*$, for any $q \in Q$, $(h(w))(q)$ is the state that we reach in A when reading word w from state q. We will identify two special elements of $\mathcal{F}_q$:

- f_0, the function mapping every state of Q to the sink state q_s;
- f_1, the function mapping the initial state q_i to the final state q_f, and mapping every other state in $Q \backslash \{q_\mathsf{i}\}$ to q_s.

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

▶ **Lemma 77.** *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 definition of A, for any state $q \neq q_\mathsf{i}$, we have $(h(\mathsf{l}))(q) = q_\mathsf{s}$, so that, as q_s is a sink state, we have $(h(w))(q) = q_\mathsf{s}$ for any w that satisfies e'. Further, by definition of A, for any state q, we have $(h(\mathsf{r}))(q) \in \{q_\mathsf{s}, q_\mathsf{f}\}$, so that, for any state q and w that satisfies e', we have $(h(w))(q) \in \{q_\mathsf{s}, q_\mathsf{f}\}$. 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. ◀

Hence, consider the query Q_b whose existence is guaranteed by Lemma 75, and such that all its possible worlds satisfy e', and construct the query $Q_\mathsf{a} := \mathrm{accum}_{h,\circ}(Q_\mathsf{b})$ – we see h as a rank-invariant accumulation map. We conclude the proof of Proposition 73 by showing that POSS is NP-hard for Q_a, even when the input po-database consists only of totally ordered po-relations; and that $|Q_\mathsf{a}(D)| \leqslant 2$ in any case:

Proof of Proposition 73. To see that Q_{a} has at most two possible results on D, observe that, for any po-database D, writing $Q(D)$ as a word $w \in \mathcal{D}_{\pm}$, we know that w matches e'. Hence, by Lemma 77, we have $h(w) \in \{f_0, f_1\}$, so that $Q_{\mathsf{a}}(D) \in \{f_0, f_1\}$.

To see that `POSS` in NP-hard for Q_{a} even on totally ordered po-relations, we reduce the balanced checking problem for Q to `POSS` for Q_{a} with the trivial reduction: we claim that for any po-database D, $Q(D)$ is balanced iff $f_1 \in Q_{\mathsf{a}}(D)$, which is proven by Lemma 77 again. Hence, $Q(D)$ is balanced iff (D, f_1) is a positive instance of `POSS`. This concludes the reduction. ◀

E.2.2 Proof of Theorem 31 for `CERT`

We rely on Proposition 73, proven in Section E.2.1. We show that it implies the part of Theorem 31 that concerns `CERT`:

Proof. Consider the query Q from Proposition 73. We show a PTIME reduction from the NP-hard problem of `POSS` for Q (for totally ordered input po-databases) to the negation of the `CERT` problem for Q (for input po-databases of the same kind). The query Q uses accumulation, so it is of the form $\mathrm{accum}_{h,\oplus}(Q')$.

Consider an instance of `POSS` for Q consisting of an input po-database D and candidate result $v \in \mathcal{M}$. Evaluate $R = Q'(D)$ in PTIME by Proposition 6, and compute in PTIME an arbitrary possible world L' of R: this can be done by a topological sort of R. Let $v' = \mathrm{accum}_{h,\oplus}(L')$. If $v = v'$ then (D, v) is a positive instance for `POSS` for Q. Otherwise, we have $v \neq v'$. Now, solve the `CERT` problem for Q on the input (D, v'). If the answer is yes, then (D, v) is a negative instance for `POSS` for Q. Otherwise, there must exist a possible world L'' in $pw(R)$ with $v'' = \mathrm{accum}_{h,\oplus}(L'')$ and $v'' \neq v'$. However, we know that $|Q(D)| \leqslant 2$ by Proposition 73. Hence, as $v \neq v'$ and $v' \neq v''$, we must have $v = v''$. So (D, v) is a positive instance for `POSS` for Q.

Thus, we have reduced `POSS` for Q in PTIME to the negation of `CERT` for Q, showing that `CERT` for Q is coNP-hard. ◀

E.3 Revisiting Section 6

For the proof of the results of Section E.3, refer to the proof of the corresponding results in Section 7: Theorem 32 is proven together with Theorem 21, Theorem 33 is proven together with Theorem 26.

F Proofs for Section 8 (Duplicate Consolidation)

F.1 Proof of Theorem 39

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

▶ **Definition 78.** 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 $id_1 \sim id_2 \Leftrightarrow T(id_1) = T(id_2)$;
- $E := \{(id'_1, id'_2) \in {ID'}^2 \mid id'_1 \neq id'_2 \wedge \exists (id_1, id_2) \in id'_1 \times id'_2 \text{ s.t. } id_1 < id_2\}$.

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

▶ **Proposition 79.** *For any po-relation* Γ*,* $\mathrm{dupElim}(\Gamma)$ *completely fails iff* G_Γ *has a cycle.*

Proof. 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 exists $id_{1i}, id_{2i} \in id'_1$ 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 L. Assume by contradiction that for all $1 \leqslant i \leqslant n$, id'_i forms an id-set of L. Let us show by induction on j that for all $1 \leqslant j \leqslant n$, $id_{21} \leqslant_L id_{2j}$. 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 proven by induction implies that $id_{21} \leqslant_L id_{2n}$, and we had $id_{2n} <_\Gamma id_{11}$ and therefore $id_{2n} <_L id_{11}$, so this contradicts the fact that id'_1 is an id-set. Thus, dupElim fails in 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 in which dupElim does not fail; this latter fact is clear by construction of L, so we must only show that L obeys the comparability relations of Γ. Now, let $t_1 < t_2$ in Γ. Either for some $1 \leqslant j \leqslant n$, $t_1, t_2 \in id'_j$ and then $t_1 <_{L_j} t_2$ by construction which means $t_1 <_L t_2$; 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 $t_1 <_L t_2$. Hence, L is a linear extension, which concludes the proof. ◀

We can now state and prove the result:

▶ **Theorem 39.** *For any po-relation Γ, we can test in PTIME if* dupElim(Γ) *completely fails; if it does not, we can compute in PTIME a po-relation Γ' such that $pw(\Gamma') =$* dupElim(Γ).

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 79, we can determine in PTIME whether dupElim(Γ) fails.

If it does not, we let $G_\Gamma = (ID', E)$ 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 Rel(Γ') is the set of all tuples within the bag Rel(Γ) (but with no duplicates).

Now, it is easy to check that $pw(\Gamma') =$ dupElim(Γ). Indeed, any possible world L of Γ' can be achieved in dupElim(Γ) by considering, as in the proof of Proposition 79, some possible world of Γ obtained following the topological sort of G_Γ defined by L. This implies that $pw(\Gamma') \subseteq$ dupElim(Γ).

Conversely, for any possible world L of Γ, dupElim(L) fails unless, for each tuple value, the occurrences of that tuple value in 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(L) can also be obtained as the corresponding linear extension of Γ'. Hence, we have dupElim(Γ) $\subseteq pw(\Gamma')$, proving their equality and concluding the proof. ◀

F.2 Possibility and Certainty Results

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 $Q(D)$, 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 PosRA$^{\mathrm{acc}}$ 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$^{\mathrm{acc}}$ queries.

F.2.1 Adapting the Results of Section 5–7

All complexity upper bounds in Sections 5–7 are proven by first evaluating the query result in PTIME using Proposition 6. So we can still evaluate the query in PTIME, using in addition Theorem 39. 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 on which to solve `POSS` and `CERT`. Hence, in what follows, we can assume that no complete failure occurs at any stage.

Now, the only assumptions that are made on the po-relation obtained from query evaluation are proven using the following facts:

- For Theorem 21 and Theorem 32, that the property of having a constant width is preserved during PosRA$_{\texttt{LEX}}$ query evaluation, using Proposition 56;
- For Theorem 26 and Theorem 33, that the property of having a constant ia-width is preserved during PosRA query evaluation, using Proposition 66.

Hence, it suffices to show the analogous preservation results for the dupElim operator. We now do so.

▶ **Proposition 80.** *For any constant $k \in \mathbb{N}$ and po-relation Γ of width $\leqslant k$, if* $\mathrm{dupElim}(\Gamma)$ *does not completely fail then it has width $\leqslant k$.*

Proof. It suffices to show that to every antichain A of $\mathrm{dupElim}(\Gamma)$ corresponds 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 $\mathrm{dupElim}(\Gamma)$ in the proof of Theorem 39 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. ◀

▶ **Proposition 81.** *For any constant $k \in \mathbb{N}$, there exists $k' \in \mathbb{N}$ such that, for any po-relation Γ of ia-width $\leqslant k$, if $\Gamma' := \mathrm{dupElim}(\Gamma)$ does not completely fail then Γ' has ia-width $\leqslant k'$.*

Proof. Let $k \in \mathbb{N}$ and fix $k' := 2^k$. Consider an ia-partition $\mathbf{P} = (c_1, \ldots, c_n)$ of minimal cardinality of Γ (hence, of cardinality $\leqslant k$). Define a partition $\mathbf{P}'$ of Γ' with classes indexed by the powerset of $\mathbf{P}$, where each element id of Γ' is mapped to the class of $\mathbf{P}'$ corresponding to the set of the classes of $\mathbf{P}$ that contain some tuple $t \in \Gamma$ which is in id. It is clear that $\mathbf{P}'$ is a partition, and that it has cardinality $\leqslant k'$. We now show that $\mathbf{P}'$ is an ia-partition of Γ'.

We first observe the following: for any class c' of $\mathbf{P}'$, either c' is a singleton class (i.e., it contains only one element in R') or the classes of $\mathbf{P}$ to which P' corresponds are all incomparable (i.e., there are no comparability relations between any elements of them in R). To see why, assume to the contrary the existence of $c' \in \mathbf{P}'$ that contains two different elements $id_1 \neq id_2$ of R' such that the subset of $\mathbf{P}$ associated to c' contains two distinct classes $c_{\mathsf{a}} \neq c_{\mathsf{b}}$ of $\mathbf{P}$ that are not incomparable: without loss of generality, we have $r_{\mathsf{a}} < r_{\mathsf{b}}$ for some tuples $r_{\mathsf{a}} \in c_{\mathsf{a}}$ and $r_{\mathsf{b}} \in c_{\mathsf{b}}$, and, by definition of classes being indistinguishable subsets, this implies that all elements of c_{a} are less than all elements of c_{b} in R. Now, the existence of id_1 and id_2 in R' implies that there are two distinct tuple values $v_1 \neq v_2$ such that there are two tuples $s_1 \neq s_2$ in c_{a} and $t_1 \neq t_2$ in c_{b} with s_1 and t_1 having value v_1 and

s_2 and t_2 having value v_2. Then we have $s_1 <_R t_2$ and $s_2 <_R t_1$ so that there is an edge in G_R from id_1 to id_2 and from id_2 to id_1. Hence G_R is not acyclic, so dupElim(R) completely fails, contradicting our assumption. Hence, our preliminary claim in proven.

The preliminary claim implies that any c' in $\mathbf{P}'$ is an antichain. Otherwise, assuming that $id_1 <_{R'} id_2$ for $id_1, id_2 \in c'$, by the preliminary claim all classes of $\mathbf{P}$ associated to c' are incomparable, but, taking $t_1 \in id_1$ and $t_2 \in id_2$ such that $t_1 <_R t_2$, t_1 and t_2 cannot be both in the same class of $\mathbf{P}$ (as they are antichains) so they are in two different classes which are associated to c' and are comparable, a contradiction.

Second, let us show that any c' in $\mathbf{P}'$ is an indistinguishable set, concluding the proof of the fact that $\mathbf{P}'$ is an ia-partition. More specifically, we must show that for any class c' of $\mathbf{P}'$ and for any two tuples $id \neq id'$ in c', for any tuple id'' of R' not in c', we have $id'' <_{R'} id$ iff $id'' <_{R'} id'$ and $id >_{R'} id''$ iff $id' >_{R'} id''$. We show the first of the four implications; the other three are symmetric. Assume that $id'' < id$; then there are $t'' \in id''$, $t \in id$ such that $t'' <_R t$. Let c be the class of $\mathbf{P}$ in which t occurs; we cannot have $t'' \in c$ as c is an antichain. As c is in the subset of $\mathbf{P}$ associated to c' and $id' \in c'$, there is $t' \in id'$ which is in c. Now, as c is indistinguishable and $t'' \notin c$, we have $t'' <_R t'$, so that $id'' < id'$. Hence, c' is an indistinguishable set. This proves that $\mathbf{P}'$ is an ia-partition, and concludes the proof. ◀

F.2.2 Proof of Theorem 40

▶ **Theorem 40.** *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 (totally ordered relation). We compute the po-relation Γ' such that $pw(\Gamma') = Q(D)$ in PTIME using Proposition 6 and the po-relation $\Gamma :=$ dupElim(Γ') in PTIME using Theorem 39. If duplicate elimination fails, we vacuously reject for `POSS` and `CERT`, following the remark at the beginning of Appendix F.2. Otherwise, the result is a po-relation Γ, with the property that each tuple value is realized exactly once, by definition of dupElim. 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 17, so we need only study the case of `POSS`, namely, decide whether $L \in pw(\Gamma)$. As L and Γ have no duplicate tuples, there is only one way to match each tuple of L to a tuple of Γ. Build Γ'' from Γ by adding, for each pair $t_i <_L t_{i+1}$ of consecutive tuples of L, the order constraint $t'_i <_{\Gamma''} t'_{i+1}$ to the corresponding tuples in Γ''. We claim that $L \in pw(\Gamma)$ iff the resulting possible world is a po-relation, i.e., its transitive closure is still antisymmetric, which can be tested in linear time 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 doesn't 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 L, and the order relations of Γ are reflected in L as it is a linear extension of Γ, so we deduce the existence of a cycle in L, a contradiction. ◀

REFERENCES FOR THE APPENDIX

BGR97 D. Bechet, P. d. Groote, and C. Retoré. A complete axiomatisation for the inclusion of series-parallel partial orders. In *RTA*, 1997.

Bv96 H. L. Bodlaender and B. van Antwerpen–de Fluiter. *Parallel algorithms for series parallel graphs*. Springer, 1996.

Dil50 R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 1950.

Ful55 D. R. Fulkerson. Note on dilworth's decomposition theorem for partially ordered sets. In *Proc. Amer. Math. Soc*, 1955.

GJ79 M. R. Garey and D. S. Johnson. *Computers And Intractability. A Guide to the Theory of NP-completeness.* W. H. Freeman, 1979.

Möh89 R. H. Möhring. Computationally tractable classes of ordered sets. In *Algorithms and Order*. Springer, 1989.

Sch03 B. Schröder. *Ordered Sets: An Introduction.* Birkhäuser, 2003.