Possible and Certain Answers for Queries over Order-Incomplete Data

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

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

Abstract

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

1998 ACM Subject Classification H.2.1 Database Management – Logical Design

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

Digital Object Identifier 10.4230/LIPIcs.TIME.2017.4

1 Introduction

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

This paper studies query evaluation over order-incomplete data in a relational setting [1]. Our running example is that of restaurants and hotels from travel websites, ranked according to proprietary functions. An example query could compute the union of ranked lists of restaurants from distinct websites, or ask for a ranked list of pairs of a restaurant and a hotel in the same district. As we do not know how the proprietary order is defined, the query result may become *uncertain*: there may be multiple reasonable orderings of restaurants in the

union result, or multiple orderings of restaurant–hotel pairs. We also study the application of order-aware *accumulation* to the query result, where each possible order may yield a different value: e.g., extracting only the highest ranked pairs, concatenating their names, or assessing the attractiveness of a district based on its best restaurants and hotels.

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

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

Main contributions. We introduce a query language with accumulation for order-incomplete data, which generalizes the positive relational algebra [1] with aggregation as the outermost operation. We define a bag semantics for this language, without assuming that a single choice of order can be made (unlike, e.g., rank aggregation [15]): we use *partial orders* to represent all orders that are consistent with the input data. We then undertake the first general study of the *complexity of possible and certain answers for queries over such data*. We show that these problems are respectively NP-complete and coNP-complete, the main difficulties being the existence of duplicate tuple values in the data and the use of order-aware accumulation. Fortunately, we can show realistic tractable cases: certainty is in PTIME without accumulation, and both problems are tractable under reasonable restrictions on the input and on the query.

The rest of this paper is organized as follows. In Section 2, we introduce our data model and our query language. We define and exemplify the problems of possible and certain answers in Section 3. We then study their complexity, first in the general case (Section 4), then in restricted settings that ensure tractability (Sections 5 and 6). We study extensions to the language, namely duplicate elimination and group-by, in Section 7. We compare our model and results with related work in Section 8, and conclude in Section 9.

Full proofs of all results are given in an extensive appendix, for lack of space. Please note that this version of the paper removes some erroneous results relative to an earlier arXiv version and the conference proceedings version: see Appendix G for details.

2 Data Model and Query Language

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

restname	*distr*
Gagnaire	8
TourArgent	5

(a) *Rest* table

hotelname	*distr*
Mercure	5
Balzac	8
Mercure	12

(b) *Hotel* table

hotelname	*distr*
Balzac	8
Mercure	5
Mercure	12

(c) $Hotel_2$ table

Figure 1 Running example: Paris restaurants and hotels

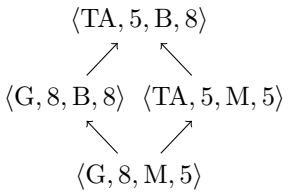

Figure 2 Example 2

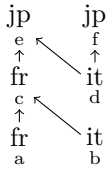

Figure 3 Example 11

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

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

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

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

Query language. We now define a bag semantics for *positive relational algebra* operators, to manipulate po-relations with queries. The positive relational algebra, written PosRA, is a standard query language for relational data [1]. We will extend PosRA later in this section with *accumulation*, and add further extensions in Section 7. Each PosRA operator applies to po-relations and computes a new po-relation; we present them in turn.

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

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

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

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

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

is incomparable for $<''$ with every $id' \in ID'$; for each $id_1, id_2 \in ID$, we have $id_1 <'' id_2$ iff $id_1 < id_2$; for each $id'_1, id'_2 \in ID'$, we have $id'_1 <'' id'_2$ iff $id'_1 <' id'_2$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Consider now $Q' := \Pi(\sigma_{\mathit{Rest.distr} = \mathit{Hotel.distr}}(Q))$, where Π projects out $\mathit{Hotel.distr}$. The possible worlds of $Q'(D)$ are $(\langle \mathrm{G}, \mathrm{B}, 8\rangle, \langle \mathrm{TA}, \mathrm{M}, 5\rangle)$ and $(\langle \mathrm{TA}, \mathrm{M}, 5\rangle, \langle \mathrm{G}, \mathrm{B}, 8\rangle)$, intuitively reflecting

two different opinions on the order of restaurant–hotel pairs in the same district. Defining Q'' similarly to Q' but replacing $\times_{\texttt{DIR}}$ by $\times_{\texttt{LEX}}$ in Q, we have $pw(Q''(D)) = (\langle \text{G}, \text{B}, 8\rangle, \langle \text{TA}, \text{M}, 5\rangle)$.

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

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

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

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

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

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

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

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

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

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

3 Possibility and Certainty

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

▶ **Definition 7 (Possibility and Certainty).** Let Q be a PosRA query, D be a po-database, and L a list relation. The *possibility problem* (`POSS`) asks if $L \in pw(Q(D))$, i.e., if L is a possible result. The *certainty problem* (`CERT`) asks if $pw(Q(D)) = \{L\}$, i.e., if L is the only possible result.

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

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

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

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

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

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

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

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

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

Accumulation can also be used for a *tuple-level comparison*. To check whether the first occurrence of a tuple t_1 precedes any occurrence of t_2, we define h_5 for all $n \in \mathbb{N}$ by $h_5(t_1,n) := \top$, $h_5(t_2,n) := \bot$ and $h_5(t,n) := \varepsilon$ for $t \neq t_1, t_2$, and a monoid operator $\oplus$ such that $\top \oplus \top = \top \oplus \bot = \top$, $\bot \oplus \bot = \bot \oplus \top = \bot$: assuming that t_1 and t_2 are both present, then the result is $\top$ if the first occurrence of t_1 precedes any occurrence of t_2, and it is $\bot$ otherwise.

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

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

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

Technical difficulties. The main challenge to solve `POSS` and `CERT` for a PosRA query Q on an input po-database D is that the tuple values of the desired result L may occur multiple times in the po-relation $Q(D)$, making it hard to match L and $Q(D)$. In other words, even though we may compute the po-relation $Q(D)$ in PTIME (by Proposition 3) and present it to the user, they still cannot easily "read" possible and certain answers out of the po-relation:

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

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

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

4 General Complexity Results

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

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

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

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

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

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

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

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

For PosRA queries, however, we show that CERT is in PTIME. As we will see later, this follows from the tractability of CERT for PosRA$^{\text{acc}}$ on *cancellative monoids* (Theorem 23).

[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.

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

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

5 Tractable Cases for POSS on PosRA Queries

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

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

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

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

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

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

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

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

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

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

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

Disallowing product. We have shown the tractability of POSS when disallowing the $\times_{\text{DIR}}$ operator, when the input po-relations are assumed to have bounded width. We now show that if we disallow both kinds of product, we obtain tractability for more general input po-relations. Specifically, we will allow input po-relations that are almost totally ordered, i.e., have bounded *width*; and we will also allow input po-relations that are almost unordered, which we measure using a new order-theoretic notion of *ia-width*. The idea of ia-width is to decompose the relation in classes of indistinguishable sets of incomparable elements:

▶ **Definition 19.** Given a poset $P = (ID, <)$, a subset $A \subseteq ID$ is an *indistinguishable antichain* if it is both an antichain (there are no $x, y \in A$ such that $x < y$) and an *indistinguishable set* (or *interval* [17]): for all $x, y \in A$ and $z \in ID \backslash A$, we have $x < z$ iff $y < z$, and $z < x$ iff $z < y$.

An *indistinguishable antichain partition* (ia-partition) of P is a partition $ID = A_1 \sqcup \cdots \sqcup A_n$ of ID such that each A_i for $1 \leqslant i \leqslant n$ is an indistinguishable antichain. The *cardinality* of the partition is n. The *ia-width* of P is the cardinality of its smallest ia-partition. The *ia-width* of a po-relation is that of its underlying poset, and the *ia-width* of a po-database is the maximal ia-width of its po-relations.

Hence, any po-relation Γ has ia-width at most $|\Gamma|$, with the trivial ia-partition consisting of singleton indistinguishable antichains, and unordered po-relations have an ia-width of 1. Po-relations may have low ia-width in practice if order is completely unknown except for a few comparability pairs given by users, or when they consist of objects from a constant number of types that are ordered based only on some order on the types.

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

▶ **Theorem 20.** *For any fixed $k \in \mathbb{N}$ and fixed $PosRA_{\text{no}\times}$ query Q, the* POSS *problem for Q is in PTIME when all po-relations of the input po-database have either ia-width $\leqslant k$ or width $\leqslant k$.*

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

▶ **Theorem 21.** *There is a $PosRA_{\text{LEX}}$ query and a $PosRA_{\text{DIR}}$ query for which the* POSS *problem is NP-complete even when the input po-database is restricted to consist only of one totally ordered and one unordered po-relation.*

6 Tractable Cases for Accumulation Queries

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

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

▶ **Definition 22 [23].** For any monoid $(\mathcal{M}, \oplus, \varepsilon)$, we call $a \in \mathcal{M}$ *cancellable* if, for all $b, c \in \mathcal{M}$, we have that $a \oplus b = a \oplus c$ implies $b = c$, and we also have that $b \oplus a = c \oplus a$ implies $b = c$. We call $\mathcal{M}$ a *cancellative monoid* if all its elements are cancellable.

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

▶ **Theorem 23.** *CERT is in PTIME for any PosRA$^{\text{acc}}$ query that performs accumulation in a cancellative monoid.*

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

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

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

▶ **Definition 24.** A PosRA$^{\text{acc}}$ query is said to perform *finite* accumulation if the accumulation monoid $(\mathcal{M}, \oplus, \varepsilon)$ is finite.

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

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

▶ **Definition 25.** Recall that the accumulation map h has in general two inputs: a tuple and its position. A PosRA$^{\text{acc}}$ query is said to be *position-invariant* if its accumulation map ignores the second input, so that effectively its only input is the tuple itself.

Note that accumulation in the monoid is still performed in order, so we can still perform, e.g., concatenation. These two restrictions do not suffice to make POSS and CERT tractable (see Appendix D.2), but we will use them to lift the results of Section 5.

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

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

▶ **Theorem 26.** *For any PosRA$^{\text{acc}}_{\text{LEX}}$ query performing* finite *accumulation, POSS and CERT are in PTIME on po-databases of bounded width.*

We can then adapt the tractability result for queries without product (Theorem 20):

▶ **Theorem 27.** *For any PosRA$^{\text{acc}}_{\text{no}\times}$ query performing* finite *and* position-invariant *accumulation, POSS and CERT are in PTIME on po-databases whose relations have either bounded width or bounded ia-width.*

The finiteness assumption is important, as the previous result does not hold otherwise. Specifically, there exists a query that performs *position-invariant* but not *finite* accumulation, for which POSS is NP-hard even on unordered po-relations (see Appendix D.4).

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

▶ **Proposition 28.** *For any PosRA query Q, the following problems are in PTIME:*

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

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

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

7 Extensions

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

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

▶ **Definition 29.** Let $(\mathcal{M}, \oplus, \varepsilon)$ be a monoid and $h : \mathcal{D}^k \to \mathcal{M}$ be an accumulation map (cf. Definition 4), and let $\mathbf{A} = A_1, ..., A_n$ be a sequence of attributes: we call $\text{accumGroupBy}_{h,\oplus,\mathbf{A}}$ an *accumulation operator with group-by*. Letting L be a list relation with compatible schema, we define $\text{accumGroupBy}_{h,\oplus,\mathbf{A}}(L)$ as an *unordered* relation that has, for each tuple value $t \in \pi_{\mathbf{A}}(L)$, one tuple $\langle t, v_t \rangle$ where v_t is $\text{accum}_{h,\oplus}(\sigma_{A_1=t.A_1,...A_n=t.A_n}(L))$ with π and σ on the list relation L having the expected semantics. The result on a po-relation Γ is the set of unordered relations $\{\text{accumGroupBy}_{h,\oplus,\mathbf{A}}(L) \mid L \in pw(\Gamma)\}$.

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

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

▶ **Theorem 30.** *There is a $\text{PosRA}^{\text{accGBy}}$ query Q with finite and position-invariant accumulation, not using $\times_{DIR}$, such that POSS for Q is NP-hard even on totally ordered po-relations.*

This result contrasts with the tractability of POSS for $\text{PosRA}_{\text{LEX}}$ queries (Theorem 15) and for $\text{PosRA}^{\text{acc}}_{\text{LEX}}$ queries with finite accumulation (Theorem 26) on totally ordered po-relations.

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

▶ **Theorem 31.** *All CERT tractability results from Section 6 extend to $\text{PosRA}^{\text{accGBy}}$ when imposing the same restrictions on query operators, accumulation, and input po-relations.*

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

▶ **Definition 32.** Given a totally ordered po-relation $(ID, T, <)$, a subset ID' of ID is an *indistinguishable duplicate set* (or *id-set*) if for every $id_1, id_2 \in ID'$, we have $T(id_1) = T(id_2)$, and for every $id \in ID \backslash ID'$, we have $id < id_1$ iff $id < id_2$, and $id_1 < id$ iff $id_2 < id$.

▶ **Example 33.** Consider the totally ordered relation $\Gamma_1 := \Pi_{hotelname}(Hotel)$, with *Hotel* as in Figure 1. The two "Mercure" tuples are not an id-set: they disagree on their ordering with "Balzac". Consider now a totally ordered relation $\Gamma_2 = (ID_2, T_2, <_2)$ whose only possible world is a list relation (A, B, B, C) for some tuples A, B, and C over $\mathcal{D}$. The set $\{id \in ID_2 \mid T_2(id) = B\}$ is an id-set in Γ_2. Note that a singleton is always an id-set.

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

▶ **Example 34.** In Example 33, we have $\mathrm{dupElim}(\Gamma_1) = \emptyset$ but $\mathrm{dupElim}(\Gamma_2) = (A, B, C)$.

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

▶ **Definition 35.** For each list relation L, we let Γ_L be a po-relation such that $pw(\Gamma_L) = \{L\}$. Letting Γ be a po-relation, we set $\mathrm{dupElim}(\Gamma) := \bigcup_{L \in pw(\Gamma)} \mathrm{dupElim}(\Gamma_L)$. We say that $\mathrm{dupElim}(\Gamma)$ *completely fails* if $\mathrm{dupElim}(\Gamma) = \emptyset$, i.e., $\mathrm{dupElim}(\Gamma_L) = \emptyset$ for every $L \in pw(\Gamma)$.

▶ **Example 36.** Consider the totally ordered po-relation *Rest* from Figure 1, and a totally ordered po-relation $Rest_2$ whose only possible world is (Tsukizi, Gagnaire). Consider $Q := \mathrm{dupElim}(\Pi_{restname}(Rest) \cup Rest_2)$. Intuitively, Q combines restaurant rankings, using duplicate consolidation to collapse two occurrences of the same name to a single tuple. The only possible world of Q is (Tsukizi, Gagnaire, TourArgent), since duplicate elimination fails in the other possible worlds: indeed, this is the only possible way to combine the rankings.

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

▶ **Theorem 37.** *For any po-relation* Γ*, we can test in PTIME if* $\mathrm{dupElim}(\Gamma)$ *completely fails; if it does not, we can compute in PTIME a po-relation* Γ' *such that* $pw(\Gamma') = \mathrm{dupElim}(\Gamma)$*.*

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

▶ **Theorem 38.** *No operator among those of PosRA and* dupElim *can be expressed through a combination of the others.*

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

▶ **Proposition 39.** *For any po-relation* Γ*, we have* $\text{dupElim}(\Gamma \cup \Gamma) = \text{dupElim}(\Gamma)$*: in particular, one completely fails iff the other does.*

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

▶ **Theorem 40.** *All* `POSS` *and* `CERT` *tractability results of Sections 4–6, except Theorem 20 and Theorem 27, extend to PosRA and* $\text{PosRA}^{\text{acc}}$ *where we allow* dupElim *(but impose the same restrictions on query operators, accumulation, and input po-relations).*

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

▶ **Theorem 41.** *For any PosRA query* Q*,* `POSS` *and* `CERT` *for* $\text{dupElim}(Q)$ *are in PTIME.*

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

8 Comparison With Other Formalisms

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

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

▶ **Proposition 42.** *For any PosRA query* Q *and a po-relation* D*,* $\text{bag}(Q(D)) = Q(\text{bag}(D))$ *where* $Q(D)$ *is defined according to our semantics and* $Q(\text{bag}(D))$ *is defined by* $BALG^1_+$*.*

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

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

Algebra on pomsets. We now compare our work to algebras defined on *pomsets* [20, 22], which also attempt to bridge partial order theory and data management (although, again, they do not study possibility and certainty). *Pomsets* are labeled posets quotiented by isomorphism (i.e., renaming of identifiers), like po-relations. A major conceptual difference between our formalism and that of [20, 22] is that their language focuses on processing *connected components* of the partial order graph, and their operators are tailored for that semantics. As a consequence, their semantics is *not* a faithful extension of bag semantics, i.e., their language would not satisfy the counterpart of Proposition 42 (see for instance the semantics of union in [20]). By contrast, we manipulate po-relations that stand for sets of possible list relations, and our operators are designed accordingly, unlike those of [20] where transformations take into account the structure (connected components) of the entire poset graph. Because of this choice, [20] introduces non-monotone operators that we cannot express, and can design a duplicate elimination operator that cannot fail. Indeed, the possible failure of our duplicate elimination operator is a direct consequence of its semantics of operating on each possible world, possibly leading to contradictions.

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

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

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

access the order relation [43, 6]. Some works also study uncertainty on totally ordered *numerical* domains [38, 39], while we look at general order relations.

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

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

9 Conclusion

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

Acknowledgements. We are grateful to Marzio De Biasi, Pálvölgyi Dömötör, and Mikhail Rudoy, from `cstheory.stackexchange.com`, for helpful suggestions. This research was partially supported by the Israeli Science Foundation (grant 1636/13) and the Blavatnik ICRC.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, 1995.
2. Bogdan Alexe, Mary Roth, and Wang-Chiew Tan. Preference-aware integration of temporal data. *PVLDB*, 8(4), 2014.
3. Lyublena Antova, Christoph Koch, and Dan Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *ICDT*. 2007.
4. Anastasios Arvanitis and Georgia Koutrika. PrefDB: Supporting preferences as first-class citizens in relational databases. *IEEE TKDE*, 26(6), 2014.
5. Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with incomplete information. *J. ACM*, 58(1), 2010.
6. Michael Benedikt and Luc Segoufin. Towards a characterization of order-invariant queries over tame graphs. *Journal of Symbolic Logic*, 74, 2009.
7. Andeas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. Posets. In *Graph Classes. A Survey*, chapter 6. SIAM, 1987.
8. Peter Buneman, Achim Jung, and Atsushi Ohori. Using powerdomains to generalize relational databases. *TCS*, 91(1), 1991.
9. Jan Chomicki and David Toman. Time in database systems. In *Handbook of Temporal Reasoning in Artificial Intelligence*. Elsevier, 2005.
10. Edgar F. Codd. Extending the database relational model to capture more meaning. *TODS*, 4(4), 1979.
11. Latha S. Colby, Edward L. Robertson, Lawrence V. Saxton, and Dirk Van Gucht. A query language for list-based complex objects. In *PODS*, 1994.
12. Latha S. Colby, Lawrence V. Saxton, and Dirk Van Gucht. Concepts for modeling and querying list-structured data. *Information Processing & Management*, 30(5), 1994.
13. Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 1950.
14. Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. Rank aggregation methods for the Web. In *WWW*, 2001.
15. Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
16. Wenfei Fan, Floris Geerts, and Jef Wijsen. Determining the currency of data. *TODS*, 37(4), 2012.
17. Roland Fraïssé. L'intervalle en théorie des relations; ses généralisations, filtre intervallaire et clôture d'une relation. *North-Holland Math. Stud.*, 99, 1984.
18. D. R. Fulkerson. Note on Dilworth's decomposition theorem for partially ordered sets. In *Proc. Amer. Math. Soc*, 1955.
19. Michael R. Garey and David S. Johnson. *Computers And Intractability. A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.
20. Stéphane Grumbach and Tova Milo. An algebra for pomsets. In *ICDT*, 1995.
21. Stéphane Grumbach and Tova Milo. Towards tractable algebras for bags. *JCSS*, 52(3), 1996.
22. Stéphane Grumbach and Tova Milo. An algebra for pomsets. *Inf. Comput.*, 150(2), 1999.
23. John M. Howie. *Fundamentals of semigroup theory*. Oxford: Clarendon Press, 1995.
24. Tomasz Imieliński and Witold Lipski. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.
25. Neil Immerman. Relational queries computable in polynomial time. *Inf. Control*, 68(1-3), 1986.
26. Marie Jacob, Benny Kimelfeld, and Julia Stoyanovich. A system for management and analysis of preference data. *VLDB Endow.*, 7(12), 2014.

27 Werner Kiessling. Foundations of preferences in database systems. In *VLDB*, 2002.
28 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.
29 Leonid Libkin. A semantics-based approach to design of query languages for partial information. In *Semantics in Databases*, 1998.
30 Leonid Libkin. Data exchange and incomplete information. In *PODS*, 2006.
31 Leonid Libkin. Incomplete data: What went wrong, and how to fix it. In *PODS*, 2014.
32 Leonid Libkin. SQL's three-valued logic and certain answers. In *ICDT*, 2015.
33 Leonid Libkin and Limsoon Wong. Query languages for bags and aggregate functions. *J. Comput. Syst. Sci.*, 55(2), 1997.
34 Witold Lipski, Jr. On semantic issues connected with incomplete information databases. *TODS*, 4(3), 1979.
35 Wilfred Ng. An extension of the relational data model to incorporate ordered domains. *TODS*, 26(3), 2001.
36 Bernd Schröder. *Ordered Sets: An Introduction*. Birkhäuser, 2003.
37 Richard T. Snodgrass, Jim Gray, and Jim Melton. *Developing time-oriented database applications in SQL*. Morgan Kaufmann, 2000.
38 Mohamed A. Soliman and Ihab F. Ilyas. Ranking with uncertain scores. In *ICDE*, 2009.
39 Mohamed A. Soliman, Ihab F. Ilyas, and Shalev Ben-David. Supporting ranking queries on uncertain and incomplete data. *VLDBJ*, 19(4), 2010.
40 Richard P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 1986.
41 Kostas Stefanidis, Georgia Koutrika, and Evaggelia Pitoura. A survey on representation, composition and application of preferences in database systems. *TODS*, 36(3), 2011.
42 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
43 Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *JCSS*, 54(1), 1997.
44 Manfred K Warmuth and David Haussler. On the complexity of iterated shuffle. *JCSS*, 28(3), 1984.

A Proofs for Section 2 (Data Model and Query Language)

A.1 Proof of Theorem 1

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

We actually prove a stronger result, namely Theorem 38, where we add the dupElim operator to PosRA operators. We consider 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 $[\leqslant n]$ constant expression, and so this query has no equivalent that does not use the $[t]$ constant expression.
- For $[\leqslant n]$, observe that $[\leqslant 2]$ is a po-relation with a non-empty order, while any query involving the other operators will have empty order (none of our unary and binary operators turns unordered po-relations into an ordered one, and the $[t]$ constant expression produces an unordered po-relation).

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

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

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

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

duplicate elimination. For any constant a not in $\mathbb{N}$, consider the po-database D_a consisting of a single unordered po-relation with name R formed of two identical unary tuples $\langle a\rangle$ and $\langle a\rangle$. Let $Q = \text{dupElim}(R)$. Then, $Q(D_a)$ is the po-relation consisting of the single tuple $\langle a\rangle$. No PosRA query without duplicate elimination has the same semantics, as no other operator than duplicate elimination can create a po-relation containing only once the constant a for any input Γ_a.

Observe that product operators are the only ones that can increase arity, so taken together they are non-redundant with the other operators. There remains to prove that each of $\times_{\texttt{DIR}}$ and $\times_{\texttt{LEX}}$ is not redundant. As in Section 5, we use the name $\text{PosRA}_{\texttt{DIR}}$ for the fragment of PosRA where $\times_{\texttt{LEX}}$ is not used; and $\text{PosRA}_{\texttt{LEX}}$ for the fragment of PosRA where $\times_{\texttt{DIR}}$.

A.1.1 Transformations Not Expressible in $\text{PosRA}_{\texttt{LEX}}$ + dupElim

We rely on Propositions 52 and 82: the result of any $\text{PosRA}_{\texttt{LEX}}$ query (possibly with dupElim), when it does not completely fail, has a *width* (see Definition 16 in Section 5) bounded by a function of the width of the original po-database. On the other hand, consider the query $Q = R \times_{\texttt{DIR}} R$ and an input po-database D_n where R is mapped to $[\leqslant n]$ (an input relation of width 1) for an arbitrary R_n. Then $Q(D_n)$ is a po-relation of width n, which shows Q is not expressible with the operators of $\text{PosRA}_{\texttt{LEX}}$ and dupElim.

A.1.2 Transformations Not Expressible in $\mathrm{PosRA}_{\mathtt{DIR}}$ + dupElim

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 the *concatenation* of po-relations:

▶ **Definition 43.** The *concatenation* $\Gamma \cup_{\mathtt{CAT}} \Gamma'$ of two po-relations Γ and Γ' is the series composition of their two partial orders. Note that $pw(\Gamma \cup_{\mathtt{CAT}} \Gamma') = \{L \cup_{\mathtt{CAT}} L' \mid L \in pw(\Gamma), L' \in pw(\Gamma')\}$, where $L \cup_{\mathtt{CAT}} L'$ is the concatenation of two list relations in the standard sense.

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

▶ **Lemma 44.** *For any arity $n \in \mathbb{N}$ and distinguished relation names R and R', there is a $PosRA_{LEX}$ query Q_n 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 $\Gamma \cup_{CAT} \Gamma'$.*

Proof. For any $n \in \mathbb{N}$ and names R and R', consider the following query (using again numerical attribute names for simplicity):

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

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

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

▶ **Lemma 45.** *For any arity $n \in \mathbb{N}_+$ and distinguished relation names R and R', there is no $PosRA_{DIR}$ query Q_n (possibly with* dupElim*) such that, for any po-relations Γ and Γ' of arity n, letting D be the po-database that maps R to Γ and R' to Γ', the query result $Q_n(D)$ is $\Gamma \cup_{CAT} \Gamma'$.*

To prove Lemma 45, we first introduce the following concept:

▶ **Definition 46.** Let $v \in \mathcal{D}$. We call a po-relation $\Gamma = (ID, T, <)$ *v-impartial* if, for any two identifiers id_1 and id_2 and $1 \leqslant i \leqslant \mathrm{a}(\Gamma)$ such that exactly one of $T(id_1).i$, $T(id_2).i$ is v, the following holds: id_1 and id_2 are *incomparable*, namely, neither $id_1 < id_2$ nor $id_2 < id_1$ hold.

▶ **Lemma 47.** *Let $v \in \mathcal{D} \backslash \mathbb{N}$ be a value. For any $PosRA_{DIR}$ query Q, possibly with* dupElim*, for any po-database D of v-impartial po-relations, the po-relation $Q(D)$ (when duplicate elimination does not completely fail) 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 singleton constant expressions, the claim is trivial as they contain less than two tuples.
- For the $[\leqslant i]$ constant expressions, the claim is immediate as $v \notin \mathbb{N}$.

We now prove the induction step:

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

- For projection, the claim is also immediate as the property to prove is maintained when reordering, copying or deleting attributes. Indeed, considering again two identifiers id'_1 and id'_2 of Γ' and $1 \leqslant i' \leqslant \mathrm{a}(\Gamma')$, the respective preimages id_1 and id_2 in Γ of id'_1 and id'_2 before the projection satisfy the same condition for some different $1 \leqslant i \leqslant \mathrm{a}(\Gamma)$ 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 $\Gamma'' := \Gamma \cup \Gamma'$, writing $\Gamma'' = (ID'', T'', <'')$, assume by contradiction the existence of two identifiers $id_1, id_2 \in \Gamma''$ and $1 \leqslant i \leqslant \mathrm{a}(\Gamma'')$ such that exactly one of $T''(id_1).i$ and $T''(id_2).i$ is v but (without loss of generality) $id_1 < id_2$ in Γ''. It is easily seen that, as id_1 and id_2 are not incomparable, they must come from the same relation; but then, as that relation was v-impartial, we have a contradiction.
- For duplicate elimination, the property is preserved as duplicate elimination (when it does not fail) results in a po-relation where the order between tuples with different values is preserved.
- 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 write $\Gamma'' = (ID'', T'', <'')$ as above. Assume that there are two identifiers id''_1 and id''_2 of ID'' and $1 \leqslant i \leqslant \mathrm{a}(\Gamma'')$ that violate the v-impartiality of Γ''. Let $(id_1, id'_1), (id_2, id'_2) \in ID \times ID'$ be the pairs of identifiers used to create id''_1 and id''_2. We distinguish on whether $1 \leqslant i \leqslant \mathrm{a}(\Gamma)$ or $\mathrm{a}(\Gamma) < i \leqslant \mathrm{a}(\Gamma) + \mathrm{a}(\Gamma')$. In the first case, we deduce that exactly one of $T(id_1).i$ and $T(id_2).i$ is v, so that in particular $id_1 \neq id_2$. Thus, by definition of the order in $\times_{\mathtt{DIR}}$, it is easily seen that, because id''_1 and id''_2 are comparable in Γ'', id_1 and id_2 must compare in the same way in Γ, contradicting the v-impartiality of Γ. The second case is symmetric. ◀

We now conclude with the proof of Lemma 45:

Proof. Let us assume by way of contradiction that there is $n \in \mathbb{N}_+$ and a $\mathrm{PosRA}_{\mathtt{DIR}}$ query Q_n, possibly with dupElim that captures $\cup_{\mathtt{CAT}}$. Let $v \neq v'$ be two distinct values in $\mathcal{D} \backslash \mathbb{N}$, and consider the singleton po-relation Γ containing one identifier of value t and Γ' containing one identifier of value t', where t (resp. t') are tuples of arity n containing n times the value v (resp. v'). Consider the po-database D mapping R to Γ and R' to Γ'. Write $\Gamma'' := Q_n(D)$. By our assumption, as $\Gamma'' = (ID'', T'', <'')$ must be $\Gamma \cup_{\mathtt{CAT}} \Gamma'$, it must contain an identifier $id \in ID''$ such that $T''(id) = t$ and an identifier $id' \in ID''$ such that $T''(id') = t'$. Now, as Γ and Γ' are (vacuously) v-impartial, we know by Lemma 47 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, we know that id and id' must be incomparable in $<''$, so there is a possible world of Γ'' where id' precedes id. This contradicts the fact that, as we should have $\Gamma'' = \Gamma \cup_{\mathtt{CAT}} \Gamma'$, the po-relation Γ'' should have exactly one possible world, namely, (t, t'). ◀

Lemma 44 and Lemma 45 conclude the proof of Theorem 1.

A.2 Proof of Proposition 3

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

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

- If Q is a relation name R, $Q(D)$ is obtained in linear time.
- If Q is a constant expression, $Q(D)$ is obtained in constant time.

Table 1 Summary of complexity results for possibility and certainty

	Query	Restrict. on accum.	Input po-relations	Complexity	
POSS	PosRA/PosRA$^{\text{acc}}$	—	arbitrary	NP-c.	(Thm. 12)
CERT	PosRA$^{\text{acc}}$	—	arbitrary	coNP-c.	(Thm. 13)
CERT	PosRA	—	arbitrary	PTIME	(Thm. 14)
POSS	PosRA$_{\text{LEX}}$	—	totally ordered	PTIME	(Thm. 15)
POSS	PosRA$_{\text{LEX}}$	—	width $\leqslant k$	PTIME	(Thm. 17)
POSS	PosRA$_{\text{DIR}}$	—	totally ordered	NP-c.	(Thm. 18)
POSS	PosRA$_{\text{no}\times}$	—	ia-width or width $\leqslant k$	PTIME	(Thm. 20)
POSS	PosRA$_{\text{LEX}}$/PosRA$_{\text{DIR}}$	—	1 total. ord., 1 unord.	NP-c.	(Thm. 21)
CERT	PosRA$^{\text{acc}}$	cancellative	arbitrary	PTIME	(Thm. 23)
POSS	PosRA$^{\text{acc}}$	finite and pos.-invar.	totally ordered	NP-c.	(Thm. 71)
CERT	PosRA$^{\text{acc}}$	finite and pos.-invar.	totally ordered	coNP-c.	(Thm. 72)
both	PosRA$^{\text{acc}}_{\text{LEX}}$	finite	width $\leqslant k$	PTIME	(Thm. 26)
both	PosRA$^{\text{acc}}_{\text{no}\times}$	finite and pos.-invar.	ia-width or width $\leqslant k$	PTIME	(Thm. 27)
POSS	PosRA$^{\text{acc}}_{\text{no}\times}$	pos.-invar.	unordered	NP-c.	(Thm. 78)

- If $Q = \sigma_\psi(Q')$ or $Q = \Pi_{k_1 \dots k_p}(Q')$, $Q(D)$ is obtained in time linear in $|Q'(D)|$, and we conclude by the induction hypothesis.
- If $Q = Q_1 \cup Q_2$ or $Q = Q_1 \times_{\text{LEX}} Q_2$ or $Q = Q_1 \times_{\text{DIR}} Q_2$, $Q(D)$ is obtained in time linear in $|Q_1(D)| \times |Q_2(D)|$ and we conclude by the induction hypothesis. ◀

B Proofs for Section 4 (General Complexity Results)

We summarize the complexity results of Sections 4–6 in Table 1.

B.1 Proofs of Theorems 12 and 13

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

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

We first show the upper bounds:

▶ **Proposition 48.** *For any PosRA or PosRA$^{\text{acc}}$ query Q,* POSS *for Q is in NP and* CERT *for Q is in co-NP.*

Proof. We show the results for PosRA$^{\text{acc}}$ queries, as the same clearly holds for PosRA queries. To show the NP membership of POSS, evaluate in PTIME the query without accumulation using Proposition 3, 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 is PTIME-evaluable), 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 show the lower bounds. We first show the lower bound of Theorem 12 for POSS on a PosRA query. In fact, when arbitrary po-relations are allowed, POSS is already hard for a trivial query: we will use non-trivial PosRA queries later to show hardness of POSS on restricted input po-relations (cf. Theorem 18 and Theorem 21).

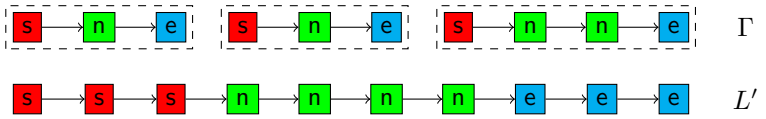

Figure 4 Example for the proof of Proposition 49, with $E = (1, 1, 2)$ and $B = 4$.

▶ **Proposition 49.** *There is a PosRA query Q such that the* POSS *problem for Q is NP-hard.*

This result can also be shown from existing work [WH84] about the complexity of the so-called *shuffle problem*: given a string w and a tuple of strings $s_1, \ldots, s_n$ on the fixed alphabet $A = \{a, b\}$, decide whether there is an interleaving of $s_1, \ldots, s_n$ which is equal to w. It is easy to see that there is a reduction from the shuffle problem to the POSS problem, by representing each string s_i as a totally ordered relation L_i of tuples labeled a and b that code the string, letting Γ be the po-relation which is the union of the L_i, and asking if the totally ordered relation that codes w is a possible world of the identity query on the po-relation Γ. Hence, as the shuffle problem is shown to be NP-hard in [WH84], this implies the same for POSS. We nevertheless give a self-contained proof of Proposition 49, because we will be extending this proof to show different results in Theorem 18. We note that our proof is in fact very similar to the hardness proof of [WH84]; see specifically Lemma 3.2 of [WH84].

Proof. The reduction is from the UNARY-3-PARTITION problem, which is NP-hard [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. We use the trivial identity query $Q := R$, where R is a relation name of arity 1. We will use an input po-database D that maps the relation name R to a po-relation Γ, and we now describe how to construct the input relation $\Gamma = (ID, T, <)$ in PTIME from the UNARY-3-PARTITION instance.

We set ID to be $\{id_i^j \mid 1 \leqslant i \leqslant 3m, 1 \leqslant j \leqslant n_i + 2\}$: this is constructible in PTIME, because the input to UNARY-3-PARTITION is written in unary. The relation Γ will have arity 1 and domain $\{\mathsf{s}, \mathsf{n}, \mathsf{e}\}$, where s, n and e are three arbitrary distinct values chosen from $\mathcal{D}$ (standing for "start", "inner", and "end"). We set $T(id_i^1) := \mathsf{s}$ and $T(id_i^{n_i+2}) := \mathsf{e}$ for all $1 \leqslant i \leqslant 3m$, and set $T(id_i^j) := \mathsf{n}$ in all other cases, i.e., for all $1 \leqslant i \leqslant 3m$ and all $2 \leqslant j \leqslant n_i + 1$. Last, we define the order relation $<$ by letting $id_i^j < id_i^{j'}$ for all $1 \leqslant i \leqslant 3m$ and $1 \leqslant j < j' \leqslant n_i + 2$. This implies in particular that, for all $1 \leqslant i, i' \leqslant 3m$, for all $1 \leqslant j \leqslant n_i + 2$ and $1 \leqslant j' \leqslant n_{i'} + 2$, if $(i, j) \neq (i', j')$, then the elements id_i^j and $id_{i'}^{j'}$ are comparable by $<$ iff $i = i'$.

Now, let L' be the list relation $\mathsf{s}^3\mathsf{n}^B\mathsf{e}^3$, where exponents denote repetition of tuples, and let L be the list relation $(L')^m$, which we will use as a candidate possible world. We now claim that the UNARY-3-PARTITION instance defined by E and B has a solution iff $L \in pw(\Gamma)$, which concludes the proof because the reduction is clearly in PTIME.

To see why the reduction is correct, we first show that, if E is a positive instance of UNARY-3-PARTITION, then there is a linear extension $<'$ of $<$ which witnesses that $L \in pw(\Gamma)$. Indeed, consider a 3-partition $\mathbf{s} = (s_1^i, s_2^i, s_3^i)$ for $1 \leqslant i \leqslant m$, with $n_{s_1^i} + n_{s_2^i} + n_{s_3^i} = B$ for all $1 \leqslant i \leqslant m$, and each integer of $\{1, \ldots, 3m\}$ occurring exactly once in $\mathbf{s}$. We can realize L from $\mathbf{s}$, picking successively the following for $1 \leqslant i \leqslant m$ to realize L': the tuples $id_1^{s_p^i}$ for $1 \leqslant p \leqslant 3$ that are mapped to s by T; the tuples $id_{j_p}^{s_p^i}$ for $1 \leqslant p \leqslant 3$ and $2 \leqslant j_p \leqslant n_{s_p^i} + 1$ that are mapped to n by T (hence, B tuples in total, by the condition on s); the tuples $id_{n_{s_p^i}+2}^{s_p^i}$

for $1 \leqslant p \leqslant 3$ that are mapped to e by T.

Conversely, we show that, if there is a linear extension $<'$ of $<$ which witnesses that $L \in pw(\Gamma)$, then we can build a 3-partition $\mathbf{s} = (s_1^i, s_2^i, s_3^i)$ for $1 \leqslant i \leqslant m$ which satisfies the conditions above. To see why, we first observe that, for each $1 \leqslant i \leqslant m$, for the i-th occurrence of the sublist L' in L, there must be three distinct values s_1^i, s_2^i, s_3^i, such that the elements of ID which occur in $<'$ at the positions of the value n in this occurrence of L' are precisely the elements of the form $id_{s_p^i}^{j_p}$ for $1 \leqslant p \leqslant 3$ and $1 \leqslant j_p \leqslant n_{s_p^i} + 1$. Indeed, we show this claim for increasing values of i, from $i = 1$ to $i = m$. For the i-th occurrence of L' for some $1 \leqslant i \leqslant m$, we define s_1^i, s_2^i, s_3^i, such that the elements s^3 in this occurrence of L' are mapped to $id_{s_1^i}^1, id_{s_2^i}^1, id_{s_3^i}^1$: they must indeed be mapped to such elements because they are the only ones mapped to s by T. Now, the elements of the form $id_{s_p^i}^{j_p}$ for $1 \leqslant p \leqslant 3$ and $1 \leqslant j_p \leqslant n_{s_p^i} + 1$ are the only ones that can be enumerated, because are the only ones that have not been enumerated yet, and they have no ancestors mapped to s by T that have not been enumerated. Further, all these elements must be enumerated, because this is the only possible way for $<'$ to be able to enumerate e-labeled elements, namely, the $id_{s_p^i}^{n_{s_p^i}+2}$ for $1 \leqslant p \leqslant 3$. Now that we have defined the 3-partition $\mathbf{s}$, it is clear by definition of a linear extension that all numbers in $\mathbf{s}$ must be distinct. Further, as $<'$ achieves L', by considering each occurrence of L', we know that, for $1 \leqslant i \leqslant m$, we have $s_1^i + s_2^i + s_3^i = B$. Hence, $\mathbf{s}$ witnesses that E is a positive instance to the UNARY-3-PARTITION problem.

This establishes the correctness of the reduction, and concludes the proof. ◀

To show the lower bound for PosRA$^{\text{acc}}$, we show a general lemma about reducing POSS and CERT for PosRA queries to the same problems for PosRA$^{\text{acc}}$ queries:

▶ **Lemma 50.** *For any arity $k \in \mathbb{N}$, there exists an infinite and cancellative monoid $(\mathcal{M}_k, \oplus, \varepsilon)$ (see Definition 22), a position-invariant and arity-k accumulation map h_k (see Definition 25), and a PTIME-evaluable accumulation operator $\text{accum}_{h_k,\oplus}$ such that, for any PosRA query Q of arity k, the POSS and CERT problems for Q are respectively equivalent to the POSS and CERT problems for the PosRA$^{\text{acc}}$ query $\text{accum}_{h_k,\oplus} Q$.*

Proof. Fix $k \in \mathbb{N}$. We will use the identity accumulation operator. Consider the monoid $(\mathcal{M}_k, \oplus, \varepsilon)$ defined as follows: $\mathcal{M}_k$ is the list relations on $\mathcal{D}^k$, that is, the finite sequences of elements of $\mathcal{D}^k$, the neutral element ε is the empty list, and the associative operation $\oplus$ is the concatenation of list relations. This clearly defines a monoid, and it is clearly cancellative. Let h_k be the position-invariant accumulation map that maps any tuple t to the singleton list relation $[t]$ containing precisely one tuple with that value.

Now, consider the query $Q' := \text{accum}_{h_k,\oplus}(Q)$. Let D be an po-database. It is clear that any list relation L is a possible world of $Q(D)$ iff L is a possible result of $Q'(D)$: in other words, we have $pw(Q(D)) = pw(Q'(D))$. This clearly ensures that POSS and CERT for Q are respectively equivalent to POSS and CERT for Q'. ◀

We deduce:

▶ **Corollary 51.** *There is a PosRA$^{\text{acc}}$ query Q such that the POSS problem for Q is NP-hard.*

What remains is to show the hardness result for CERT and PosRA$^{\text{acc}}$. This result is more complex, and is presented (in a slightly stronger form) as Theorem 72 in Appendix D.2.

B.2 Proof of Theorem 14

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

Proof. Let Q be the PosRA query of interest, and let $k \in \mathbb{N}$ be its arity: let $(\mathcal{M}_k, \oplus)$ be the cancellative monoid (see Definition 22) and h_k be the accumulation map obtained from Lemma 50. By Lemma 50, we know that CERT for Q is equivalent to CERT for the $\text{PosRA}^{\text{acc}}$ query $Q' := \text{accum}_{h,\oplus}$, which is clearly constructible in PTIME.

Now, by Theorem 23 (proven in Appendix D.1), we know that the CERT problem is in PTIME for Q', because it performs accumulation in a cancellative monoid. Hence, using the PTIME reduction above, we deduce that the CERT problem for Q is in PTIME as well. ◀

C Proofs for Section 5 (Tractable Cases for POSS on PosRA Queries)

C.1 Totally Ordered Inputs

C.1.1 Tractability Result: Proof of Theorems 15 and 17

The point of restricting to $\text{PosRA}_{\text{LEX}}$ 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 52.** *Let $k \geqslant 2$ and Q be a $\text{PosRA}_{\text{LEX}}$ 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 $\text{PosRA}_{\text{LEX}}$ 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 (singletons and constant chains) have width 1.

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 $\Gamma := \Gamma_1 \times_{\text{LEX}} \Gamma_2$ has bound $k_1 \cdot k_2$. To show this, consider any set A of Γ containing strictly more than $k_1 \cdot k_2$ identifiers, which we see as pairs of an identifier of Γ_1 and an identifier 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 identifiers 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 $\times_{\texttt{LEX}}$ 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 identifiers 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 $\times_{\texttt{LEX}}$, 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 $\text{PosRA}_{\texttt{LEX}}$ queries when the input po-database consists of relations of bounded width. We now prove Theorem 17, which clearly generalizes Theorem 15. We will prove both the result for $\text{PosRA}_{\texttt{LEX}}$ queries and its extension to $\text{PosRA}^{\text{acc}}_{\texttt{LEX}}$ queries with finite accumulation (Theorem 26).

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

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

Let $\Gamma := Q(D)$ be the po-relation obtained by evaluating the query Q of interest on the input po-database D, excluding the accumulation operator if any (so we are evaluating a $\text{PosRA}_{\texttt{LEX}}$ query). We can compute this in PTIME using Proposition 3. Letting k' be the constant (only depending on Q and k) given by Proposition 52, we know that $w(\Gamma) \leqslant k'$.

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

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

▶ **Theorem 53.** *For any constant $k' \in \mathbb{N}$, and accumulation operator $\text{accum}_{h,\oplus}$ with finite domain, we can compute in PTIME, for any input po-relation Γ such that $w(\Gamma) \leqslant k'$, the set $\text{accum}_{h,\oplus}(\Gamma)$.*

Indeed, by what precedes, we can assume that the query has already been evaluated to a po-relation; further, once the possible results are determined, it is immediate to solve possibility and certainty.

To take care of this task, we need the following notions:

▶ **Definition 54.** A *chain partition* of a poset P is a partition $\mathbf{\Lambda} = (\Lambda_1, \ldots, \Lambda_n)$ of the elements of P, i.e., $P = \Lambda_1 \sqcup \cdots \sqcup \Lambda_n$, such that each Λ_i is a total order. (However, P may feature comparability relations not present in the Λ_i, i.e., relating elements in Λ_i to elements in Λ_j for $i \neq j$.) The *width* of the partition $\mathbf{\Lambda} = (\Lambda_1, \ldots, \Lambda_n)$ is n.

▶ **Definition 55.** 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 56 [Dil50].** *Any poset P has a chain partition of width $w(P)$.*

▶ **Theorem 57 [Ful55].** *For any poset P, we can compute in PTIME a chain partition of P of minimal width.*

We now prove Theorem 53:

Proof of Theorem 53. Consider a po-relation $\Gamma = (ID, T, <)$, with underlying poset $P = (ID, <)$. Using Theorems 56 and Theorem 57, compute in PTIME a chain partition $\mathbf{\Lambda}$ of P of width k'. For $1 \leqslant i \leqslant k'$, write $n_i := |\Lambda_i|$, and for $0 \leqslant j \leqslant n_i$, write $\Lambda_i^{\leqslant j}$ to denote the subset of Λ_i containing the first j elements of the chain (in particular $\Lambda_i^{\leqslant 0} = \emptyset$).

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 $\bigsqcup_{i=1}^{k'} \Lambda_i^{\leqslant m_i}$.

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 Λ_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 Λ_i at a later position, and because Λ_i is a chain it must include all elements before, hence, $S \cap \Lambda_i = \Lambda_i^{\leqslant m_i}$. As $\mathbf{\Lambda}$ is a chain partition 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:

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

where the operator "$-$" is the component-by-component integer difference on tuples, 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 $\Lambda_i[m_i]$ 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 $\Lambda_i[m_i]$ was not actually maximal because it was comparable to (and smaller than) some $\Lambda_j[m_j]$ 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. ◀

$\mathbf{PosRA}_{\mathtt{LEX}}$ 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 17 for queries with no accumulation.

Recall that the `CERT` problem is in PTIME for such queries by Theorem 14, 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 53:

▶ **Theorem 58.** *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 list relation L, whether $L \in pw(\Gamma)$.*

Proof. The proof of Theorem 53 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 $t((0, \ldots, 0)) := \text{true}$, 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 53. ◀

This concludes the proof of Theorem 17, and, as an immediate corollary, of Theorem 15.

C.1.2 Hardness result: Proof of Theorem 18

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

Note that, unlike Proposition 49, this result does not follow immediately from the results of [WH84]. Remember that [WH84] studies the *shuffle problem* which asks, given a string w and a tuple of strings $s_1, \ldots, s_n$, whether there is an interleaving of $s_1, \ldots, s_n$ which is equal to w. It is easy to describe the possible interleavings of the s_i in PosRA as a union of totally

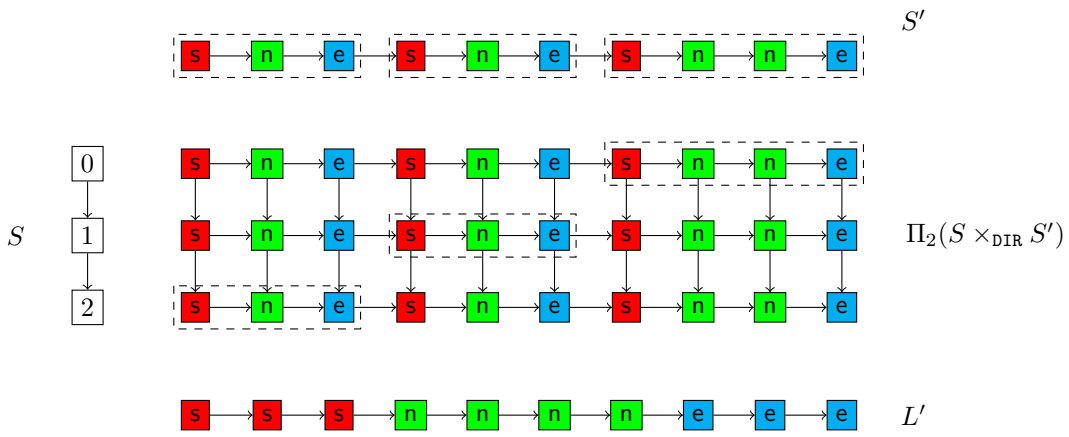

Figure 5 Example for the proof of Theorem 18, with $E = (1, 1, 2)$ and $B = 4$. The dashed parts of the grid represent T, as mentioned in the proof sketch.

ordered list relations, but it is more challenging to test, *with a constant query*, whether the s_i have an interleaving equal to w. This is what we do in the proof of Theorem 18:

Proof. The proof is an adaptation of Proposition 49. Again, we reduce from the NP-hard 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. We fix $\mathcal{D} := \mathbb{N} \sqcup \{\mathsf{s}, \mathsf{n}, \mathsf{e}\}$, with s, n and e standing for *start*, *inner*, and *end* as in the previous proof.

Let S be the totally ordered po-relation $[\leqslant 3m - 1]$, and let S' be the totally ordered po-relation whose one possible world is 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 $Q := \Pi_2(S \times_{\mathtt{DIR}} S')$, where Π_2 projects to the attribute coming from relation S'. See Figure 5 for an illustration, and note the similarity with Figure 4. Note that S and S' are *input* relations, not constant expressions that would give the same relation.

We define the candidate possible world as follows:

- L_1 is a list relation defined as the concatenation, for $1 \leqslant i \leqslant 3m$, of $3m - i$ copies of the following sublist: one tuple with value s, n_i tuples with value n, and one tuple with value e.
- L_2 is a list relation defined as above, except that $3m - i$ is replaced by $i - 1$.
- L' is the list relation defined as in the proof of Proposition 49, namely, 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 po-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: the intuition is to eliminate parts of the grid that match to L_1 and L_2, so that we are left with an order relation that allows us to re-use the proof of Proposition 49.

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; letting S_i be the multiset of the n_j for $1 \leqslant j \leqslant l_i$, we call S_i the bag of *row-i consumed integers*. The *row-i remaining integers* are $E \backslash S_i$ (seeing E as a multiset, and performing difference of multisets by subtracting the multiplicities in S_i to the multiplicities in E).

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 proved.

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

A similar argument shows that for any linear extension L'' of 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 the elements $n_1, \ldots, n_{3m}$. So the unmatched elements when having read L_1 (at the beginning) and L_2 (at the end) are formed of $3m$ lists, 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 $3m$ lists (one for each row, with that of row i being s, n_i times n and e, for all i) and there are no order relations across sequences. 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 lists, 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 s for u_1^j, e for $u_{n_j+2}^j$, and 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. This claim is shown exactly as in the proof of Proposition 49, as L' is the same as in that proof, and T is the same order relation as Γ in that proof. This concludes the proof of the desired result. ◀

C.2 Disallowing Product

C.2.1 Tractability Result: Proof of Theorem 20

▶ **Theorem 20.** *For any fixed $k \in \mathbb{N}$ and fixed $PosRA_{\text{no}\times}$ query Q, the* POSS *problem for Q is in PTIME when all po-relations of the input po-database have either ia-width $\leqslant k$ or width $\leqslant k$.*

We start by making a simple observation:

▶ **Lemma 59.** *Any PosRA query Q without any product can be rewritten as a union of projections of selections of a constant number of input relations and constant relations.*

Proof. This follows from the fact that, for the semantics that we have defined for operators, the following is clear: selection commutes with union, selection commutes with projection, and projection commutes with union. Hence, we can perform the desired rewriting. ◀

We can thus rewrite the input query using this lemma. The idea is that we will evaluate the query in PTIME using Proposition 3, argue that the width bounds are preserved using Proposition 52, and compute a chain partition of the relations using Theorem 56 and Theorem 57. However, we first need to show analogues of Proposition 52, Theorem 56, and Theorem 57 for the new notion of ia-width. We first show the analogue of Proposition 52 for the case without product:

▶ **Proposition 60.** *Let $k \geqslant 2$ and Q be a $PosRA_{\text{no}\times}$ query. Let $k' := \max(k, q) \times |Q|$, where $|Q|$ denote the number of symbols of Q, and where q denotes the largest value such that $[\leqslant q]$ appears in Q. For any po-database D of ia-width $\leqslant k$, the po-relation $Q(D)$ has ia-width $\leqslant k'$.*

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

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

For the induction step:

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

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

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

▶ **Proposition 61.** *The ia-width of any poset, and a corresponding ia-partition, can be computed in PTIME.*

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

▶ **Lemma 62.** *For any poset $(ID, <)$ and indistinguishable antichain A, for any $A' \subseteq A$, then A' is an indistinguishable antichain.*

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

▶ **Lemma 63.** *For any poset $(ID, <)$ and indistinguishable antichains $A_1, A_2 \subseteq ID$ such that $A_1 \cap A_2 \neq \emptyset$, the union $A_1 \cup A_2$ is an indistinguishable antichain.*

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

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

We can now show Proposition 61:

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

Now assume that there is an ia-partition $\mathbf{A}'$ of cardinality $m < n$. There has to be a class A' of $\mathbf{A}'$ which intersects two different classes $A_1 \neq A_2$ of the greedy ia-partition $\mathbf{A}$, otherwise $\mathbf{A}'$ would be a refinement of $\mathbf{A}$ so we would have $m \geqslant n$. Now, by Lemma 63, $A \cup A_1$ and $A \cup A_2$, and hence $A \cup A_1 \cup A_2$, are indistinguishable antichains. By Lemma 62, this implies that $A_1 \cup A_2$ is an indistinguishable antichain. Now, when constructing the greedy ia-partition $\mathbf{A}$, the algorithm has considered one element of A_1 and one element of A_2, attempted to merge the classes A_1 and A_2, and, since it has not merged them in $\mathbf{A}$, the union $A_1 \cup A_2$ cannot be an indistinguishable antichain. We have reached a contradiction, so we cannot have $m < n$, which concludes the proof. ◀

We have shown the preservation of ia-width bounds through selection, projection, and union (Proposition 60), and shown how to compute an ia-partition in PTIME (Proposition 61). Let us now return to the proof of Theorem 20. We use Lemma 59 to rewrite the query to a union of projection of selections. We evaluate the selections and projections in PTIME by Proposition 3. As union is clearly associative and commutative, we evaluate the union of relations of width $\leqslant k$, yielding Γ, and the union of those of ia-width $\leqslant k$, yielding Γ'. The first result Γ has bounded width thanks to Proposition 52, and we can compute a chain partition of it in PTIME using Theorem 56 and Theorem 57. The second result has bounded ia-width thanks to Proposition 60, and we can compute an ia-partition of it in PTIME using Proposition 61.

Queries with no accumulation. We first prove Theorem 20 for the case without accumulation. It suffices to show the following:

▶ **Proposition 64.** *For any constant $k \in \mathbb{N}$, we can determine in PTIME, for any input po-relation Γ with width $\leqslant k$, input po-relation Γ' with ia-width $\leqslant k$, and list relation L, whether $L \in pw(\Gamma \cup \Gamma')$.*

Before proving this, we show a weaker result that restricts to a bounded-ia-width input relation:

▶ **Proposition 65.** *For any constant $k \in \mathbb{N}$, we can determine in PTIME, for any po-relation Γ with ia-width $\leqslant k$ and list relation L, whether $L \in pw(\Gamma)$.*

Proof. Let $\mathbf{A} = (A_1, \ldots, A_k)$ be an ia-partition of width k of $\Gamma = (ID, T, <)$, which can be computed in PTIME by Proposition 61. We assume that the length of the candidate possible world L is $|ID|$, as we can trivially reject otherwise.

If there is a way to realize L as a possible world of Γ, For any linear extension $<'$ of Γ, we call the *finishing order* $<'$ the permutation π of $\{1, \ldots, k\}$ obtained by considering, for each class A_i of $\mathbf{A}$, the largest position $1 \leqslant n_i \leqslant |ID|$ in $<'$ to which an element of A_i is mapped, and sorting the class indexes by ascending finishing order. We say we can realize L with finishing order π if there is a linear extension of Γ that realizes L and whose finishing order is π. Hence, it suffices to check, for every possible permutation π of $\{1, \ldots, k\}$, whether L can be realized from Γ with finishing order π: this does not make the complexity worse because

the number of finishing orders depends only on k and not on Γ, so it is constant. (Note that the order relations across classes may imply that some finishing orders are impossible to realize altogether.)

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

When reading a value t from L, consider all open classes. If none of these classes have an unused element with value t, reject, i.e., conclude that we cannot realize L as a possible world of Γ with finishing order π. Otherwise, take the open class that comes first in the finishing order, and use an arbitrary suitable element from it. Update the class to be *exhausted* if it is: in this case, check that the class was the next one in the finishing order π (and reject otherwise), and update from *blocked* to *open* the classes that must be. Once L has been completely read, accept: as $|L| = |ID|$ we know that all elements are now used.

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

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

justified the existence of a linear extension that enumerates minimal identifiers until a later position; so, reapplying the rewriting argument, we will deduce the existence of a greedy linear extension. So it only remains to show that $<''$ satisfies the order constraints of Γ.

Let us assume by way of contradiction that $<''$ violates an order constraint of Γ. There are two possible kinds of violation. The first kind is if $<'$ enumerates an element id'' between id and id' for which $id < id''$, so that having $id'' <'' id$ in $<''$ is a violation. The second kind is if $<'$ enumerates an element id'' between id and id' for which $id'' < id'$, so that having $id'' <'' id'$ in $<''$ is a violation. The second kind of violation cannot happen because we know that id' is in an open class when $<'$ considers id, i.e., we have ensured that id' can be enumerated instead of id. Hence, we focus on violations of the first kind. Consider id'' such that $id <' id'' <' id'$ and let us show that we do not have $id < id''$. Letting A'' be the class of id'', we assume that $A'' \neq A$, as otherwise there is nothing to show because the classes are antichains. Now, we know from $<'$ that we do not have $id' <' id''$, and that the class A' of id' is not exhausted when $<'$ enumerates id''. As $<'$ respects the finishing order π, and A' comes before A in π, we know that A is not exhausted either when $<'$ enumerates id''. Letting id_A be an element of A which is still unused when $<'$ enumerates id'', we know that we do not have $id_A < id''$. So as $id'' \notin A$ we know by indistinguishability that we do not have $id < id''$ either. This is what we wanted to show, so id'' cannot witness a violation of the first kind. Hence $<''$ does not violate the order constraints of Γ, and repeating this rewriting argument shows that there is a greedy linear extension that the greedy algorithm will find, contradicting the proof. ◀

We now extend this proof to show Proposition 64:

Proof. As in the proof of Proposition 65, we will enumerate all possible finishing orders for the classes of Γ', of which there are constantly many, and apply an algorithm for each finishing order π, with the algorithm succeeding iff it succeeds for some finishing order.

We first observe that if there is a way to achieve L as a possible world of $\Gamma \cup \Gamma'$ for a finishing order π, then there is one where the subsequence of the tuples that are matched to Γ' are matched following a greedy strategy as in Proposition 65. This is simply because L must then be an interleaving of a possible world of Γ and a possible world of Γ', and a match for the possible world of Γ' can be found as a greedy match, by what was shown in the proof of Proposition 65. So it suffices to assume that the tuples matched to Γ' are matched following the greedy algorithm of Proposition 65.

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

This justifies that we can solve the problem with a dynamic algorithm again. The state contains the position $\mathbf{b}$ in each chain of Γ, and a position i in the candidate possible world. As in the proof of Theorem 53, we filter the configurations so that they are sane with respect to the order constraints between the chains of Γ. For each state, we will store a Boolean value indicating whether the prefix of length i of L can be realized by $\Gamma \cup \Gamma'$ such that the tuples of Γ that are matched is the order ideal $s(\mathbf{b})$ described by $\mathbf{b}$, and such that the other tuples

of the prefix are matched to Γ' following a greedy strategy with finishing order π. By our second remark above, when the Boolean is true, the state of Γ' is uniquely determined, and we also store it as part of the state (it is polynomial) so that we do not have to recompute it each time.

From each state we can make progress by consuming the next tuple from the candidate possible world, increasing the length of the prefix, and reaching one of the following states: either match the tuple to a chain of Γ, in which case we make progress in one chain and the consumed tuples in Γ' remain the same; or make progress in Γ', in which case we look at the previous state of Γ' that was stored and consume a tuple from Γ' following the greedy algorithm of Proposition 65: more specifically, we find an unused tuple with the right label which is in the open class that appears first in the finishing order, if the class is now exhausted we verify that it was supposed to be the next one according to the finishing order, and we update the open, exhausted and blocked status of the classes.

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

$\mathbf{PosRA}^{\mathrm{acc}}_{\mathrm{no}\times}$ queries with finite and position-invariant accumulation. We now prove the result for the case of a query with accumulation. In this setting, the results for `POSS` and `CERT` follow from the following claim:

▶ **Theorem 66.** *For any constant $k \in \mathbb{N}$, and position-invariant accumulation operator $\mathrm{accum}_{h,\oplus}$ with finite domain, we can compute in PTIME, for any input po-relation Γ with width $\leqslant k$ and input po-relation Γ' with ia-width $\leqslant k$, the set $\mathrm{accum}_{h,\oplus}(\Gamma \cup \Gamma')$.*

Proof. We use Theorem 56 and Theorem 57 to compute in PTIME a chain partition of Γ, and we use Proposition 61 to compute in PTIME an ia-partition $A_1 \sqcup \cdots \sqcup A_n$ of minimal cardinality of Γ', with $n \leqslant k$.

We then apply a dynamic algorithm whose state consists of:

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

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

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

proof of Theorem 53, and it clearly considers all possible ideals of Γ' except that we identify ideals that only differ by elements in the same class which are mapped to the same value by h (but this choice does not matter because the class is an antichain and these elements are indistinguishable outside the class).

As in the proof of Theorem 53, we can ensure that all accumulation operations are in PTIME, using PTIME-evaluability of the accumulation operator, up to the technicality of storing at each state, for each of the possible accumulation results, a witnessing totally ordered relation from which to compute it in PTIME. ◀

C.2.2 Hardness result: Proof of Theorem 21

▶ **Theorem 21.** *There is a PosRA$_{\texttt{LEX}}$ query and a PosRA$_{\texttt{DIR}}$ query for which the* POSS *problem is NP-complete even when the input po-database is restricted to consist only of one totally ordered and one unordered po-relation.*

The proof is by adapting the proof of Theorem 18. The argument is exactly the same, except that we take relation S to be *unordered* rather than totally ordered. Intuitively, in Figure 5, this means that we drop the vertical edges. The proof adapts, because it only used the fact that $t'_j < t'_k$ for $j < k$ within a row-i; we never used the comparability across groups.

D Proofs for Section 6 (Tractable Cases for Accumulation Queries)

D.1 Cancellative monoids

▶ **Theorem 23.** *CERT is in PTIME for any PosRA$^{\text{acc}}$ query that performs accumulation in a cancellative monoid.*

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

▶ **Definition 67.** 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 68.** *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 Λ of P such that element x is enumerated at position p in Λ, 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 Λ 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 69.** *For any fixed (PTIME-evaluable) accumulation operator* $\mathrm{accum}_{h,\oplus}$ *we can determine in PTIME, given a po-relation* Γ*, whether* Γ *has safe swaps with respect to* h*.*

Proof. Consider each pair (id_1, id_2) of elements of Γ, of which there are quadratically many. Check in PTIME whether they are incomparable. If yes, compute in PTIME $\mathrm{pr}_\Gamma(id_1, id_2)$, and consider each pair p, $p+1$ of consecutive integers (there are linearly many). For each such pair, compute $h(T(id_1), p) \oplus h(T(id_2), p+1)$ and $h(T(id_2), p) \oplus h(T(id_1), p+1)$, and check whether are equal.

We must only argue that these expressions can be evaluated in PTIME, but this follows from the PTIME-evaluability of the accumulation operator. Specifically, to evaluate, e.g., $h(T(id_1), p) \oplus h(T(id_2), p+1)$, we build in PTIME from Γ a list relation L with $p+1$ tuples that are all labeled with the neutral element of the monoid of h except the two last ones which are labeled respectively with $T(id_1)$ and $T(id_2)$. We then evaluate the accumulation operator in PTIME on L and obtain the desired value. ◀

Now it is easily seen that Theorem 23 is implied by the following claim.

▶ **Proposition 70.** *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 3, and we can test in PTIME by Lemma 69 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 70. 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:

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

We use Lemma 68 to 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 list 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 list relation in the list is L'_2, and the last list 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 70, which, as we argued, concludes the proof of Theorem 23.

D.2 Other restrictions on accumulation

We show the additional claim that assuming finiteness and position-invariance of accumulation does not suffice to make POSS or CERT tractable. Specifically, we show the following two results:

▶ **Theorem 71.** *There is a PosRA$^{\text{acc}}$ query performing finite and position-invariant accumulation for which* POSS *is NP-hard even assuming that the input po-database contains only totally ordered po-relations.*

▶ **Theorem 72.** *There is a PosRA$^{\text{acc}}$ query performing finite and position-invariant accumulation for which* CERT *is coNP-hard even assuming that the input po-database contains only totally ordered po-relations.*

We will first show the result about POSS (Theorem 71), and then use it to show the result about CERT (Theorem 72).

D.2.1 Proof of Theorem 71 for POSS

We show the following strengthening of Theorem 71, which will be useful to prove the result for CERT in Appendix D.2.2.

▶ **Proposition 73.** *There is a PosRA$^{\text{acc}}$ query Q_{a} with finite and position-invariant accumulation such that the* POSS *problem is NP-hard for Q_{a}, 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 $|Q_{\mathsf{a}}(D)| \leqslant 2$.*

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 $Q'_{\mathsf{b}}(R,T) := [\mathsf{l}] \cup_{\mathtt{CAT}} ((R \cup T) \cup_{\mathtt{CAT}} [\mathsf{r}])$, i.e., $Q'_{\mathsf{b}}(R,T)$ is the parallel composition of R and T, preceded by l and followed by r. Recall the definition of $\cup_{\mathtt{CAT}}$ (Definition 43), and recall from Lemma 44 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 unary list relation defined by mapping each letter of w to the corresponding letter in $\mathcal{D}_-$. We define Γ^-_w as the totally ordered po-relation with $pw(\Gamma^-_w) = \{L^-_w\}$. We claim the following:

▶ **Lemma 76.** *For any $w \in \mathcal{D}^*_+$ and unary po-relation T over $\mathcal{D}_+$, we have $w \in pw(T)$ iff $\{R \mapsto \Gamma^-_w, T \mapsto T\}$ is a positive instance to the balanced checking problem for Q'_{b}; in other words, iff $Q'_{\mathsf{b}}(\Gamma^-_w, T)$ has some balanced possible world.*

Proof. For the first direction, assume that w is indeed a possible world L of T and let us construct a balanced possible world L' of $Q'_{\mathsf{b}}(\Gamma^-_w, T)$. L' starts with l. Then, L' successively contains alternatively one tuple from Γ^-_w (in their total order) and one from T (taken in the order of the linear extension that yields L). Finally, L' ends with r. L' is clearly balanced.

For the converse direction, observe that a balanced possible world of $Q'_{\mathsf{b}}(\Gamma^-_w, T)$ must consist of first l, last r, and, between the two, tuples alternatively enumerated from Γ^-_w from one of the possible worlds of T, with that possible world of T achieving w. ◀

We now use Lemma 76 to prove Lemma 75:

Proof of Lemma 75. By Theorem 18 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 Γ^-_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 (and hence Theorem 71) 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 S 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 $q_\perp$.

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

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

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

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

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

▶ **Lemma 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 $(h(\mathsf{l}))(q) = q_\bot$, so that, as $q_\bot$ is a sink state, we have $(h(w))(q) = q_\bot$ for any w that satisfies e'. Further, by definition of A, for any state q, we have $(h(\mathsf{r}))(q) \in \{q_\bot, q_\mathsf{f}\}$, so that, for any state q and w that satisfies e', we have $(h(w))(q) \in \{q_\bot, q_\mathsf{f}\}$. 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 position-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 $Q_\mathsf{b}(D)$ 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_b to POSS for Q_a with the trivial reduction: we claim that for any po-database D, there is a balanced possible world in $Q_\mathsf{b}(D)$ iff $f_1 \in Q_\mathsf{a}(D)$, which is proved by Lemma 77 again. Hence, $Q_\mathsf{b}(D)$ is balanced iff (D, f_1) is a positive instance of POSS for Q_a. This concludes the reduction. ◀

This concludes the proof of Proposition 73, hence of Theorem 71.

D.2.2 Proof of Theorem 72 for CERT

We prove Theorem 72 by relying on Proposition 73, proven in Appendix D.2.1:

Proof of Theorem 72. Consider the query Q_{a} from Proposition 73. We show a PTIME reduction from the NP-hard problem of POSS for Q_{a} (for totally ordered input po-databases) to the negation of the CERT problem for Q_{a} (for input po-databases of the same kind). The query Q_{a} uses accumulation, so it is of the form $\mathrm{accum}_{h,\oplus}(Q')$.

Consider an instance of POSS for Q_{a} consisting of an input po-database D and candidate result $v \in \mathcal{M}$. Evaluate $R = Q'(D)$ in PTIME by Proposition 3, 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_{a}. Otherwise, we have $v \neq v'$. Now, solve the CERT problem for Q_{a} on the input (D, v'). If the answer is YES, then (D, v) is a negative instance for POSS for Q_{a}. Otherwise, there must exist a possible world L'' in $pw(R)$ with $v'' = \mathrm{accum}_{h,\oplus}(L'')$ and $v'' \neq v'$. However, we know that $|pw(Q_{\mathsf{a}}(D))| \leqslant 2$ 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_{a}.

Thus, we have reduced POSS for Q_{a} in PTIME to the negation of CERT for Q_{a}, showing that CERT for Q_{a} is coNP-hard. ◀

D.3 Revisiting Section 5

For the proof of the results of this paragraph, refer to the proof of the corresponding results in Section 6: Theorem 26 is proven together with Theorem 17 in Appendix C.1.1, and Theorem 27 is proven together with Theorem 20 in Appendix C.2.1.

D.4 Hardness Without the Finiteness Assumption

We show the additional claim that POSS for $\mathrm{PosRA}^{\mathrm{acc}}_{\mathrm{no}\times}$ queries can be hard if we do not assume finiteness. Namely, we show:

▶ **Theorem 78.** *There is a position-invariant accumulation operator* $\mathrm{accum}_{h,\oplus}$ *such that* POSS *is NP-hard for the* $\mathrm{PosRA}^{\mathrm{acc}}_{\mathrm{no}\times}$ *query* $Q := \mathrm{accum}_{h,\oplus}(\Gamma)$ *(i.e., accumulation applied directly to an input po-relation* Γ*), even on input po-databases where* Γ *is restricted to be an unordered relation.*

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

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

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

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

Conversely, assume that the POSS problem has a solution. Consider a witness total order of Γ_S; it must a (possibly empty) sequence of tuples corresponding to a subset S_1 of S, then a tuple -1, then a (possibly empty) sequence corresponding to $S_2 \subseteq S$, then a tuple -1, then a (possibly empty) sequence corresponding to $S'_1 \subseteq S$, with $S = S_1 \sqcup S'_1 \sqcup S_2$. Let s_1, s'_1 and s_2 respectively be the sums of these subsets of S. The result of accumulation is then $g_{s_1} \circ f \circ g_{s_2} \circ f \circ g_{s'_1}$, which simplifies to $g_{s_1+s'_1-s_2}$. Hence, we have $s_1 + s'_1 = s_2$, so that $(S_1 \sqcup S'_1)$ and S_2 are a partition witnessing that S is a positive instance of the partition problem.

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

D.5 Other definitions

▶ **Proposition 28.** *For any PosRA query Q, the following problems are in PTIME:*

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

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

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

Proof. To solve each problem, we first compute the po-relation $\Gamma := Q(D)$ in PTIME by Proposition 3. We now address each problem in turn.

select-at-k: Considering the po-relation $\Gamma = (ID, T, <)$, we can compute in PTIME, for every element $id \in ID$, its *earliest index* $\mathsf{i}^-(id)$, which is its number of ancestors by $<$ plus one, and its *latest index* $\mathsf{i}^+(id)$, which is the number of elements of Γ minus the number of descendants of id. It is easily seen that for any element $id \in ID$, there is a linear extension of Γ where id appears at position $\mathsf{i}^-(id)$ (by enumerating first exactly the ancestors of id), or at position $\mathsf{i}^+(id)$ (by enumerating first everything except the descendants of id), or in fact at any position of $[\mathsf{i}^-(id), \mathsf{i}^+(id)]$, the *interval* of id (this is by enumerating first the ancestors of id, and then as many elements as needed that are incomparable to id, along a linear extension of these elements).

Hence, select-at-k possibility for tuple t and position k can be decided by checking, for each $id \in ID$ such that $T(id) = t$, whether $k \in [\mathsf{i}^-(id), \mathsf{i}^+(id)]$, and answering YES iff we can find such an id. For select-at-k certainty, we answer NO iff we can find an $id \in ID$ such that $k \in [\mathsf{i}^-(id), \mathsf{i}^+(id)]$ but we have $T(id) \neq t$.

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

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

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

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

E Proofs for Section 7 (Extensions)

E.1 Proof of Theorem 30: Hardness of POSS with Group-By

▶ **Theorem 30.** *There is a PosRA$^{\text{accGBy}}$ query Q with finite and position-invariant accumulation, not using $\times_{DIR}$, such that POSS for Q is NP-hard even on totally ordered po-relations.*

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

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

In the accumulation operator, the accumulation map h maps each tuple t to its second component. Further, we define the finite monoid $\mathcal{M}$ to be the *syntactic monoid* [Pin97] of the language defined by the regular expression $\mathsf{s}(\mathsf{l}_+\mathsf{l}_-|\mathsf{l}_-\mathsf{l}_+)^*\mathsf{e}$, where s (for "start"), l_- and l_+, and e (for "end") are fresh values from $\mathcal{D}$: this monoid ensures that, for any non-empty word w on the alphabet $\{\mathsf{s}, \mathsf{l}_-, \mathsf{l}_+, \mathsf{e}\}$ that starts with s and ends with e, the word w evaluates to ε in $\mathcal{M}$ iff w matches this regular expression.

We reduce from the NP-hard 3-SAT problem: we are given a conjunction of clauses $C_1, \ldots, C_n$, with each clause being a disjunction of three literals, namely, a variable or negated variable among $x_1, \ldots, x_m$, and we ask whether there is a valuation of the variables such that the clause is true. We fix an instance of this problem. We assume without loss of generality that the instance has been preprocessed to ensure that no clause contained two occurrences of the same variable (neither with the same polarity nor with different polarities).

We define the relation R to be $[\leqslant m + 3]$. The totally ordered relations S_1, S_2, and S_3 consist of $3m + 2n$ tuple values, which we define in a piecewise fashion:

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

- The third coordinate is s for all these tuples.

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

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

  The intuition for the variable choice gadget is that, for each group, we have two incomparable elements, one labeled I_- and one labeled I_+. Hence, any linear extension must choose to enumerate one after the other, committing to a valuation of the variables in the 3-SAT instance; to achieve the candidate possible world, the linear extension will then have to continue enumerating the elements of this group in the correct order.
- For the tuples with positions from $2m+1$ to $2m+2n$ (the "clause check" gadget), for each $1 \leqslant j \leqslant n$, letting $j' := 2n+j+1$, we describe tuples j' and $j'+1$ in S_1, S_2, S_3:
  - The first coordinate is $j+2$;
  - The second coordinate carries values in $\{a, b, c\}$, where we write clause C_j as $\pm x_a \vee \pm x_b \vee \pm x_c$. Specifically:
    - Value a is assigned to tuple $j'+1$ in relation S_1 and tuple j' in relation S_2;
    - Value b is assigned to tuple $j'+1$ in relation S_1 and tuple j' in relation S_2;
    - Value c is assigned to tuple $j'+1$ in relation S_1 and tuple j' in relation S_2;
  - The third coordinate carries values in $\{\mathsf{I}_-, \mathsf{I}_+\}$; namely, writing C_j as above:
    - Tuple $j'+1$ in relation S_1 carries I_+ if variable x_a occurs positively in C_j, and I_- otherwise; tuple j' in relation S_2 carries the other value;
    - The elements at the same positions in relation S_2 and S_3, respectively in S_3 and S_1, are defined in the same way depending on the sign of x_b, respectively of x_c.

  The intuition for the clause check gadget is that, for each $1 \leqslant j \leqslant n$, the tuples at levels j' and $j'+1$ check that clause C_j is satisfied by the valuation chosen in the variable choice gadget. Specifically, if we consider the order constraints on the two elements from the same group (i.e., second coordinate) which are implied by the order chosen for this variable in the variable choice gadget, the construction ensures that these order constraints plus the comparability relations of the chains imply a cycle (that is, an impossibility) iff the clause is violated by the chosen valuation.
- For the tuples with positions from $2n+2m+1$ to $3n+2m$ (the "closing gadget"), the definition is like the opening gadget but replacing e by s, namely:
  - The first coordinate is $m+3$ for all tuples in S_1 and 0 for all tuples in S_2 and S_3 (which again do not join with R);
  - The second coordinate is i for the i-th tuple in S_1;
  - The third coordinate is e for all these tuples.

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

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

To do so, we first describe the result of evaluating $\Gamma := Q'(R, S_1, S_2, S_3)$ on the relations described above. Intuitively, it is just like $\Pi_{2,3}(\sigma_{2 \neq \text{“0”}}(S_1 \cup S_2 \cup S_3))))$, but with the following

additional comparability relations: all tuples in all chains whose first coordinate carried a value i are less than all tuples in all chains whose first coordinate carried a value $j > i$. In other words, we add comparability relations across chains as we move from one "first component" value to the next. The point of this is that it forces us to enumerate the tuples of the chains in a way that "synchronizes" across all chains whenever we change the first component value. Observe that, in keeping with Proposition 52, the width of Γ has a constant bound, namely, it is 3.

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

- For the start gadget, enumerate all tuples of S_1 in the prescribed order. Hence, the current accumulation result in all n groups is s.
- For the variable choice gadget, for all i, enumerate the i-th tuples of S_1 and S_2 of the gadget in an order depending on $\nu(x_i)$: if $\nu(x_i)$ is 1, enumerate first the tuple of S_1 and then the tuple of S_2, and do the converse if $\nu(x_i) = 0$. Hence, for all $1 \leqslant i \leqslant n$, the current accumulation result in group i is $\mathsf{s}\mathsf{l}\_\mathsf{l}\_+$ if $\nu(x_i)$ is 1 and $\mathsf{s}\mathsf{l}\_+\mathsf{l}\_$ otherwise.
- For the clause check gadget, we consider each clause in order, for $1 \leqslant j \leqslant n$, maintaining the property that, for each group $1 \leqslant i \leqslant n$, the current accumulation result in group i is of the form $\mathsf{s}(\mathsf{l}\_\mathsf{l}\_+)^*$ if $\nu(x_i) = 1$ and $\mathsf{s}(\mathsf{l}\_+\mathsf{l}\_)^*$ otherwise.
  Fix a clause C_j, let $j' := 2n+j+1$ as before, and study the tuples j' and $j'+1$ of S_1, S_2, S_3. As C_j is satisfied under ν, let x_d be the witnessing literal (with $d \in \{a, b, c\}$), and let d' be the index (in $\{1, 2, 3\}$) of variable d. Assume that x_d occurs positively; the argument is symmetric if it occurs negatively. By definition, we must have $\nu(x_d) = 1$, and by construction tuple j' in relation $S_{1+(d'+1 \bmod 3)}$ must carry value $\mathsf{l}\_$ and it is in group d. Hence, we can enumerate it and group d now carries a value of the form $\mathsf{s}(\mathsf{l}\_\mathsf{l}\_+)^*\mathsf{l}\_$. Now, letting x_e be the $1 + (d' + 1 \bmod 3)$-th variable of $\{x_a, x_b, x_c\}$, the two elements of group e (tuple $j' + 1$ of $S_{1+(d'+1 \bmod 3)}$ and tuple j' of $S_{1+(d'+1 \bmod 3)}$) both had all their predecessors enumerated; so we can enumerate them in the order that we prefer to satisfy the condition on the accumulation values; then we enumerate likewise the two elements in the remaining group in the order that we prefer, and last we enumerate the second element of group d; so we have satisfied the invariants.
- Last, for the end gadget, we enumerate all tuples of S_1 and we have indeed obtained the desired accumulation result.

This concludes the proof of the forward direction.

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

- First, all tuples of the start gadget.
- Then, all tuples of the variable choice gadget. We use this to define a valuation ν: for each variable x_i, we set $\nu(x_i) = 1$ if the tuple of S_1 in group i was enumerated before the one in group S_2, and we set $\nu(x_i) = 0$ otherwise.
- Then, for each $1 \leqslant j \leqslant n$, in order, tuples $2n + j + 1$ of S_1, S_2, S_3.
  Observe that this implies that, whenever we enumerate such tuples, it must be the case that the current accumulation value for any variable x_i is of the form $\mathsf{s}(\mathsf{l}\_\mathsf{l}\_+)^*$ if $\nu(x_i) = 1$, and $\mathsf{s}(\mathsf{l}\_+\mathsf{l}\_)^*$ otherwise. Indeed, fixing $1 \leqslant i \leqslant n$, assume that we are in the first case (the second one is symmetric). In this case, the accumulation state for x_i after the variable choice gadget was $\mathsf{s}\mathsf{l}\_\mathsf{l}\_+$, and each pair of levels in the clause check gadget made us enumerate either ε (variable x_i did not occur in the clause) or one of l_l_+ or $\mathsf{l}\_+\mathsf{l}\_$

(variable x_i occurred in the clause); as the 3-SAT instance was preprocessed to ensure that each variable occurred only at most once in each clause, this case enumeration is exhaustive. Hence, the only way to obtain the correct accumulation result is to always enumerate $|\_|_+$, as if we ever do the contrary the accumulation result can never satisfy the regular expression that it should satisfy.

- Last, all tuples of the end gadget.

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

E.2 Proof of Theorem 31: Tractability of CERT with Group-By

▶ **Theorem 31.** *All CERT tractability results from Section 6 extend to PosRA*$^{\text{accGBy}}$ *when imposing the same restrictions on query operators, accumulation, and input po-relations.*

We show the following auxiliary result:

▶ **Proposition 79.** *For any PosRA*$^{\text{accGBy}}$ *query* $Q := \text{accumGroupBy}_{h,\oplus,P}Q'$ *and family* $\mathcal{D}$ *of po-databases, the CERT problem for* Q *on input po-databases from* $\mathcal{D}$ *reduces in PTIME to the CERT problem for* $\text{accum}_{h,\oplus}R$ *(where* Γ *is a po-relation name), on the family* $\mathcal{D}'$ *of po-databases mapping the name* Γ *to a subset of a po-relation of* $\{Q'(D) \mid D \in \mathcal{D}\}$.

Proof. To prove that, consider an instance of CERT for Q, defined by an input po-database D of $\mathcal{D}$ and candidate possible world L. We first evaluate $\Gamma' := Q'(D)$ in PTIME. Now, for each tuple value t in $\pi_P(\Gamma')$, let Γ_t be the restriction of Γ' to the elements matching this value; note that the po-database mapping R to Γ_t is indeed in the family $\mathcal{D}'$. We solve CERT for each $\text{accum}_{h,\oplus}\Gamma_t$ in PTIME with the candidate possible world obtained from L by extracting the accumulation value for that group, and answer YES to the original CERT instance iff all these invocations answer YES. As this process is clearly in PTIME, we must show correctness of the relation.

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

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

This allows us to show Theorem 31 by considering all results of Section 6 in turn, and showing that they extend to PosRA$^{\text{accGBy}}$ queries, under the same restrictions on operators, accumulation, and input po-relations:

- Theorem 23 extends, because `CERT` is tractable on any family $\mathcal{D}'$ of input po-databases, so tractability for PosRA$^{\text{accGBy}}$ holds for any family $\mathcal{D}$ of input po-databases.
- Theorem 26 extends, because, for any family $\mathcal{D}$ of po-databases whose po-relations have width at most k for some $k \in \mathbb{N}$, we know by Proposition 52 that the result $Q'(D)$ for $D \in \mathcal{D}$ also has width depending only on Q' and on k, and we know that restricting to a subset of $Q'(D)$ (namely, each group) does not increase the width (this is like the case of selection in the proof of Proposition 52). Hence, the family $\mathcal{D}'$ also has bounded width.
- Theorem 27 extends because we know (see Lemma 59 and subsequent observations) that the result $Q'(D)$ for $D \in \mathcal{D}$ is a union of a po-relation of bounded width and of a po-relation with bounded ia-width. Restricting to a subset (i.e., a group), this property is preserved (as in the case of selection in the proof of Proposition 52 and of Proposition 60), which allows us to conclude.

E.3 Proof of Theorems 37 and 38 and Proposition 39

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

▶ **Definition 80.** 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 81.** *For any po-relation* Γ, $\mathrm{dupElim}(\Gamma)$ *completely fails iff* G_Γ *has a cycle.*

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

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

We can now state and prove the result:

▶ **Theorem 37.** *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* $pw(\Gamma') = \mathrm{dupElim}(\Gamma)$*.*

Proof. We first observe that G_Γ can be constructed in PTIME, and that testing that G_Γ is acyclic is also done in PTIME. Thus, using Proposition 81, we can determine in PTIME whether dupElim(Γ) fails.

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

Now, it is easy to check that $pw(\Gamma') = \mathrm{dupElim}(\Gamma)$. Indeed, any possible world L of Γ' can be achieved in dupElim(Γ) by considering, as in the proof of Proposition 81, some possible world of Γ obtained following the topological sort of G_Γ defined by L. This implies that $pw(\Gamma') \subseteq \mathrm{dupElim}(\Gamma)$.

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

▶ **Theorem 38.** *No operator among those of PosRA and* dupElim *can be expressed through a combination of the others.*

Proof. This is shown in the proof of Theorem 1 in Appendix A.1. ◀

We also use the value-equality quotient to show:

▶ **Proposition 39.** *For any po-relation* Γ*, we have* $\mathrm{dupElim}(\Gamma \cup \Gamma) = \mathrm{dupElim}(\Gamma)$*: in particular, one completely fails iff the other does.*

Proof. Let G_Γ be the value-equality quotient of Γ and G'_Γ be the value-equality quotient of $\Gamma \cup \Gamma$. It is easy to see that these two graphs are identical: any edge of G_Γ witnesses the existence of the same edge in G'_Γ, and conversely any edge in G'_Γ must correspond to a comparability relation between two tuples of one of the copies of Γ (and also in the other copy, because they are two copies of the same relation), so that it also witnesses the existence of the same edge in Γ. Hence, one duplicate elimination operation completely fails iff the other does, because this is characterized by acyclicity of the value-equality quotient (see Proposition 81). Further, by Theorem 37, as duplicate elimination is constructed from the value-equality quotient, we have indeed the equality that we claimed. ◀

E.4 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$^{\text{acc}}$ query extended with dupElim, we likewise say that its possible accumulation results are $\emptyset$.

This implies that for any PosRA query Q extended with dupElim, for any input po-database D, and for any candidate possible world v, the POSS and CERT problems for Q are vacuously false on instance (D, v) if complete failure occurs at any stage when evaluating $Q(D)$. The same holds for PosRA$^{\text{acc}}$ queries.

E.4.1 Proof of Theorem 40: Adapting the Results of Section 4–6

▶ **Theorem 40.** *All* POSS *and* CERT *tractability results of Sections 4–6, except Theorem 20 and Theorem 27, extend to PosRA and PosRA*$^{\text{acc}}$ *where we allow* dupElim *(but impose the same restrictions on query operators, accumulation, and input po-relations).*

All complexity upper bounds in Sections 4–6 are proved by first evaluating the query result in PTIME using Proposition 3. So we can still evaluate the query in PTIME, using in addition Theorem 37. 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, except Theorems 20 and Theorem 27, the only assumptions that are made on the po-relation obtained from query evaluation are proved using the following facts:

- For all theorems in Section 4, for Theorem 23, and for Proposition 28, no assumptions are made, so the theorems continue to hold.
- For Theorem 17 and Theorem 26, that the property of having a constant width is preserved during PosRA$_{\text{LEX}}$ query evaluation, using Proposition 52.

Hence, Theorem 40 follows from the following width preservation result:

▶ **Proposition 82.** *For any constant $k \in \mathbb{N}$ and po-relation Γ of width $\leqslant k$, if* dupElim(Γ) *does not completely fail then it has width $\leqslant k$.*

Proof. It suffices to show that to every antichain A of dupElim(Γ) 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 dupElim(Γ) in the proof of Theorem 37 as obtained from the topological sorts of the value-equality quotient G_Γ of Γ, as $t_1 < t_2$ implies that (id_1, id_2) is an edge of G_Γ, we conclude that we have $id_1 < id_2$ in A, contradicting the fact that it is an antichain. ◀

We conclude by illustrating that Theorem 20 cannot be adapted as-is, because the preservation result that it uses does not adapt to the dupElim operator.

▶ **Example 83.** Fix $n \in \mathbb{N}$. Consider the totally ordered relation $R := [\leqslant n]$; it has width 1. Consider the po-relation $S = (ID, T, <)$ that consists of n pairwise incomparable identifiers $id_1^{\uparrow}, \ldots, id_n^{\uparrow}$ whose images by T are respectively $1, \ldots, n$, and n pairwise incomparable identifiers $id_1^{\downarrow}, \ldots, id_n^{\downarrow}$ with pairwise distinct fresh values, with the order relation $id_i^{\uparrow} < id_j^{\downarrow}$ for all $1 \leqslant i, j \leqslant n$; The po-relation S has ia-width 2, with the partition $(\{id_i^{\uparrow} \mid 1 \leqslant i \leqslant n\}, \{id_i^{\downarrow} \mid 1 \leqslant i \leqslant n\})$. Hence, $R \cup S$ would satisfy the hypotheses of Theorem 20. However, $R' :=$ dupElim$(R \cup S)$ is the po-relation consisting of tuples $id'_1, \ldots, id'_n$ with

values respectively $1, \ldots, n$, tuples $id''_1, \ldots, id''_n$ with the values of the $id_i^{\downarrow}$, and the order relation $id'_i < id'_j$ iff $i < j$ and $id'_i < id'_j$ for all $1 \leqslant i, j \leqslant n$.

We now observe that, for every partition of R' into two sets, there is a comparability relation going from one set to the other. Hence, R' cannot be written as the union of two non-empty po-relations. Yet, R' has width n, as witnessed by the id''_i, and it has ia-width n, as witnessed by the id'_i.

This illustrates that, when performing duplicate consolidation on the union of a constant-width po-relation and of a constant-ia-width po-relation, we cannot hope that the result has constant width, or constant ia-width, or can be written as the union of two relations where each has one of these properties.

E.4.2 Proof of Theorem 41: POSS and CERT After Removing Duplicates

▶ **Theorem 41.** *For any PosRA query Q,* `POSS` *and* `CERT` *for* dupElim(Q) *are in PTIME.*

Proof. Let D be an input po-relation, and L be the candidate possible world (a list relation). We compute the po-relation Γ' such that $pw(\Gamma') = Q(D)$ in PTIME using Proposition 3 and the po-relation $\Gamma := \text{dupElim}(\Gamma')$ in PTIME using Theorem 37. If duplicate elimination fails, we vacuously reject for `POSS` and `CERT`, following the remark at the beginning of Appendix E.4. 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 14, so we need only study the case of `POSS`, namely, decide whether $L \in pw(\Gamma)$. Let Γ_L be a po-relation that represents L. As Γ_L and Γ have no duplicate tuples, there is only one way to match each identifier of Γ_L to an identifier of Γ. Build Γ'' from Γ by adding, for each pair $id_i <_L id_{i+1}$ of consecutive tuples of Γ_L, the order constraint $id''_i <'' id''_{i+1}$ on the corresponding identifiers in Γ''. We claim that $L \in pw(\Gamma)$ iff the resulting Γ'' is a po-relation, i.e., its transitive closure is still antisymmetric, which can be tested in PTIME by computing the strongly connected components of Γ'' and checking that they are all trivial.

To see why this works, observe that, if the result Γ'' is a po-relation, it is a total order, and so it describes a way to achieve L as a linear extension of Γ because it 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 Γ_L, and the order relations of Γ are reflected in Γ_L as it is a linear extension of Γ, so we deduce the existence of a cycle in Γ_L, which is impossible by construction. Hence, we have reached a contradiction, and we deduce the desired result. ◀

E.5 Alternative Semantics for Duplicate Elimination

A main downside of our proposed semantics for dupElim is the fact that complete failure is allowed. We conclude 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 84.** Letting $A \neq B$ be two tuples, let us consider a po-relation Γ_L representing the list relation $L := (A, B, B, A)$. With weak duplicate elimination, we would have $\text{dupElim}(\Gamma_L) = (A, B, A)$.

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 85.** 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 $\mathrm{dupElim}(\Gamma) = \{(A, B, A), (A, B)\}$ 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 $\mathrm{dupElim}(L)$ for a list relation 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 86.** Applying *aggressive* dupElim to Γ_L from Example 84 yields $\{(A, B), (B, A)\}$.

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

▶ **Example 87.** Consider a po-relation Γ_L representing the list relation $L := (A, C, B, C, A)$ with distinct tuples A, B, C. Then $\mathrm{dupElim}(\Gamma_L)$ is $\{(A, C, B), (A, B, C), (B, C, A), (C, B, A)\}$. No po-relation Γ satisfies $pw(\Gamma) = \mathrm{dupElim}(L)$, 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 because some of the six permutations of $\{A, B, C\}$ are not possible worlds.

We leave for future work the question of designing a practical semantics for duplicate consolidation that can be incorporated in our framework and avoids failure.

F Proofs for Section 8 (Comparison With Other Formalisms)

▶ **Proposition 42.** *For any PosRA query Q and a po-relation D,* $\mathrm{bag}(Q(D)) = Q(\mathrm{bag}(D))$ *where $Q(D)$ is defined according to our semantics and $Q(\mathrm{bag}(D))$ is defined by $BALG^1_+$.*

Proof. There is an exact correspondence in terms of the output bags between additive union and our union; between cross product and $\times_{\mathtt{DIR}}$ and $\times_{\mathtt{LEX}}$ (both our product operations yield the same bag as output, for any input); between our selection and that of BALG^1_+, and similarly for projection (as noted before the statement of Proposition 42 in the main text, a technical subtlety is that the projection of $BALG$ can only project on a single attribute, but one can encode "standard" projection on multiple attributes). The proposition follows by induction on the query structure. ◀

We formally prove that the output of a PosRA query can be arbitrary:

▶ **Proposition 88.** *For any po-relation Γ, there is a PosRA query Q with no inputs s.t. $Q() = \Gamma$.*

To prove the result, we will need the notion of a *realizer* of a poset:

▶ **Definition 89.** [Sch03] Letting $P = (V, <)$ be a poset, we say that a set of total orders $(V, <_1), \ldots, (V, <_n)$ is a *realizer* of P if for every $x, y \in V$, we have $x < y$ iff $x <_i y$ for all i.

We will use this notion for the following lemma. This lemma is given as Theorem 9.6 of [Hir55], see also [Øre62]; we rephrase it in our vocabulary, and for convenience we also give a self-contained proof.

▶ **Lemma 90.** *Let $n \in \mathbb{N}$, and let $(P, <_P)$ be a poset that has a realizer $(L_1, \ldots, L_n)$ of size n. Then P is isomorphic to a subset Γ' of $\Gamma = [\leqslant l] \times_{\mathtt{DIR}} \cdots \times_{\mathtt{DIR}} [\leqslant l]$, with n factors in the product, for some integer $l \in \mathbb{N}$ (the order on Γ' being the restriction on that of Γ).*

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

We are now ready to prove Proposition 88:

Proof of Proposition 88. We first show that for any poset $(P, <)$, there exists a $\text{PosRA}_{\mathtt{DIR}}$ query Q such that the tuples of $\Gamma' := Q()$ all have unique values and the underlying poset of Γ' is $(P, <)$. Indeed, we can take d to be the *order dimension* of P, which is necessarily finite [Sch03], and then by definition P has a realizer of size d. By Lemma 90, there is an integer $l \in \mathbb{N}$ such that $\Gamma'' := [\leqslant l] \times_{\mathtt{DIR}} \cdots \times_{\mathtt{DIR}} [\leqslant l]$ (with n factors in the product) has a subset S isomorphic to $(P, <)$. Hence, letting ψ be a tuple predicate such that $\sigma_\psi(\Gamma'') = S$ (which can clearly be constructed by enumerating the elements of S), the query $Q' := \sigma_\psi(\Gamma'')$ proves the claim, with Γ'' expressed as above.

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

G Discussion of Changes in this Version

In the process of preparing a journal version of this paper [ABDS18], we have discovered a flaw in the proof of some of our tractability results on ia-width. We have accordingly removed these results from [ABDS18] and from the present version of this paper. However, the results still survive in the first version of this paper on arXiv [ABDS17b] and in the published version in the TIME proceedings [ABDS17a]. In this appendix, we list the affected theorems, point out the source of the error, and discuss our current understanding of their correctness.

Affected theorems. The affected theorems are numbered as follows in the TIME proceedings version [ABDS17a] and in the main text of [ABDS17b]:

- Theorem 19: tractability of `POSS` for any PosRA query on po-databases of unordered po-relations.
- Theorem 22: tractability of `POSS` for any PosRA query on po-databases of bounded-ia-width po-relations.
- Theorem 30: tractability of `POSS` and `CERT` for any $\text{PosRA}^{\text{acc}}$ query on po-databases of bounded-ia-width po-relations.

Source of the error. The error is in Proposition 66 of [ABDS17b]. This proposition claims that, for any PosRA query Q and $k \in \mathbb{N}$, there is a bound $k' \in \mathbb{N}$ such that, for any po-database D of po-relations of ia-width $\leqslant k$, the po-relation $Q(D)$ has ia-width $\leqslant k'$. The proof is by induction, but in the case of the product operators $\times_{\mathtt{LEX}}$ and $\times_{\mathtt{DIR}}$, the argument

Figure 6 Illustration of the Hasse diagram of $Q_{\texttt{LEX}}(D_n)$ in Example 91

does not correctly reflect the behavior of the product operators. For this reason, the proof of the proposition is incorrect, and this affects the theorems listed previously, because their proofs rely on Proposition 66.

Status of the results. It is easy to see that the *statement* of Proposition 66 fails to hold:

▶ **Example 91.** For any $n \in \mathbb{N}$, consider the po-relation $\Gamma_n = (ID_n, T_n, <_n)$ with $ID_n = \{1, \ldots, n\}$, with T_n being the identity function, and with $<_n$ being empty. As Γ_n is unordered, it has ia-width 1. Consider the PosRA query $Q_{\texttt{LEX}} := R \times_{\texttt{LEX}} [\leqslant 2]$. Call D_n the po-database interpreting relation name R by Γ_n, and let $\Gamma_{n,\texttt{LEX}} := Q_{\texttt{LEX}}(D_n)$. The set of identifiers of $\Gamma_{n,\texttt{LEX}}$ is $\{(i, j) \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant 2\}$ (where we use 1 and 2 as the identifiers of the tuples in $[\leqslant 2]$), and the order relation $<$ is defined as follows (see Figure 6 for an illustration):

- For all $1 \leqslant i \leqslant n$, we have $(i, 1) < (i, 2)$;
- For all $i \neq j$ in $\{1, \ldots, n\}$, for all $p, q \in \{1, 2\}$, the tuples (i, p) and (j, q) are incomparable.

We now show that the ia-width of $\Gamma_{n,\texttt{LEX}}$ is equal to $2n$, by arguing that there is no indistinguishable antichain containing two different identifiers. Indeed, consider any two identifiers $(i, p) \neq (j, q)$, assume that there is an indistinguishable antichain A that contains both of them, and let us show a contradiction. If $i = j$, then the identifiers are comparable, so they cannot both occur in A, contradicting our assumption. Otherwise, letting $p' := 3 - p$, we know that (i, p) and (i, p') are comparable, so (i, p') cannot be in A. We now see that (i, p') violates indistinguishability for A: we know that it is comparable to (i, p), but it is not comparable to (j, q) because $i \neq j$. Hence, we have a contradiction, and (i, p) and (j, q) cannot both occur in A. So indeed the ia-width of $\Gamma_{n,\texttt{LEX}}$ is equal to $2n$.

Hence, we have an example of a PosRA query using only the $\times_{\texttt{LEX}}$ product for which the query result on a po-database of ia-width 1 can have unbounded ia-width. This contradicts the statement of Proposition 66.

We note that we can also use the $\times_{\texttt{DIR}}$ product instead of $\times_{\texttt{LEX}}$, e.g., with the query $Q_{\texttt{DIR}} := R \times_{\texttt{DIR}} [\leqslant 2]$ and with the same construction: it is easy to see that $\Gamma_{n,\texttt{DIR}} := Q_{\texttt{DIR}}(D_n)$ is exactly equal to $\Gamma_{n,\texttt{LEX}}$. Hence, Proposition 66 fails even when restricted to $\text{PosRA}_{\texttt{LEX}}$ or to $\text{PosRA}_{\texttt{DIR}}$, which concludes the example.

We can also show the following result, which contradicts Theorem 19 and Theorem 22 under the assumption that P is different from NP:

▶ **Theorem 92.** *There is a PosRA query Q for which the* POSS *problem is NP-complete even when the input po-database is restricted to consist only of unordered po-relations.*

As for Theorem 30, we do not know whether a corresponding intractability result can be shown, i.e., whether we can adapt Theorem 92 to perform accumulation in a *finite* monoid rather than in the free monoid. We also note that the query used to prove Theorem 92 will use both $\times_{\texttt{DIR}}$ and $\times_{\texttt{LEX}}$, so we do not know whether a restriction of Theorem 19 or Theorem 22 to $\text{PosRA}_{\texttt{LEX}}$ or $\text{PosRA}_{\texttt{DIR}}$ could hold.

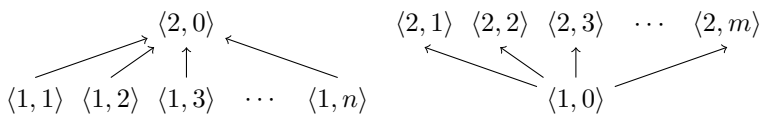

Figure 7 Illustration of the Hasse diagram of Γ'_R and Γ'_S in the proof of Theorem 92

We will show Theorem 92 in the rest of this appendix. Let $\mathsf{a} \neq \mathsf{b}$ be two distinguished domain values of $\mathcal{D}$. We will reduce from an NP-hard problem on so-called ab-*bipartite* po-relations:

▶ **Definition 93.** Let $\Gamma = (ID, T, <)$ be a po-relation. We say that Γ is *bipartite* if we can partition $ID = U \sqcup V$ such that, for any pair $id < id'$ of comparable identifiers, we have $id \in U$ and $id \in V$. (Equivalently, the Hasse diagram of the poset $(ID, <)$ is a directed bipartite graph.) We say that Γ is ab-*bipartite* if the partition can be chosen as $U := \{id \in ID \mid T(id) = \mathsf{a}\}$ and $V := \{id \in ID \mid T(id) = \mathsf{b}\}$. Note that, in this case, the domain of Γ is necessarily $\{\mathsf{a}, \mathsf{b}\}$, and the partition can be computed in PTIME simply by looking at the element labels.

We show hardness of POSS on ab-bipartite po-relations for a specific kind of possible worlds:

▶ **Proposition 94.** *The following problem is NP-hard: given an* ab*-bipartite po-relation* Γ *with partition* $U \sqcup V$*, and two integers* $0 \leqslant p \leqslant |U|$ *and* $0 \leqslant q \leqslant |V|$*, decide whether the totally ordered relation* $L_{p,q} = \mathsf{a}^p\mathsf{b}^q\mathsf{a}^{|U|-p}\mathsf{b}^{|V|-q}$ *on* $\{\mathsf{a}, \mathsf{b}\}$ *is a possible world of* Γ*.*

Proof. We reduce from the NP-hard *k-clique problem*: given an undirected graph $G = (X, E)$ and an integer $k \in \mathbb{N}$, decide whether G contains a clique of k vertices. Given the undirected graph G and the integer k, we construct the po-relation Γ by creating one a-labeled identifier in U for each vertex of X (that we identify to the vertex), creating one b-labeled identifier in V for each edge of X (that we identify to the edge), and defining the order as follows: for any edge $e = \{x, y\}$ of E, we set $x < e$ and $y < e$. It is immediate that Γ is indeed ab-bipartite. We set $p := k$ and set $q := \binom{k}{2}$. The construction is clearly in PTIME.

Now, to show correctness, if G contains a k-clique $X' \subseteq X$, we achieve the totally ordered relation $L_{p,q}$ by first enumerating all the p identifiers of X' (they are a-labeled so they are incomparable and have no ancestors), then enumerating the q edges of the clique between the vertices of X' (they are b-labeled, so incomparable, and their ancestors are all in X' so they have already been enumerated), then enumerating all remaining vertices (they are a-labeled, so incomparable and have no ancestors) and edges (they are b-labeled, so incomparable, and their ancestors have already been enumerated).

Conversely, assume that there is a topological sort of Γ that achieves $L_{p,q}$. We define X' to contain the vertices that were enumerated to achieve the prefix a^p. We know that, afterwards, we have enumerated q identifiers that were b-labeled, and the corresponding edges must have been between vertices of X', otherwise the order constraints prevent us from enumerating them. So the induced subgraph of G on X' contains q edges, i.e., it is a clique. This concludes the correctness proof and establishes NP-hardness of our problem. ◀

We are now ready to prove Theorem 92:

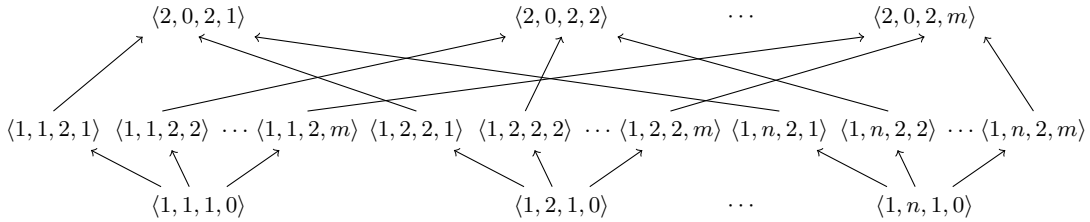

Figure 8 Illustration of the Hasse diagram of Γ' (omitting tuple $\langle 2, 0, 1, 0\rangle$) in the proof of Theorem 92

Proof. We will reduce from the NP-hard problem of Proposition 94. We start by describing formally the construction used in the reduction, i.e., the fixed query and input unordered po-relations, but the reader may find it more informative to digest the query bottom-up by reading the explanation of query evaluation given at the beginning of the correctness proof.

The fixed query is as follows:

$$Q := W \cup \Pi\left(\sigma_{=}\left(\sigma_R\left([\leqslant 2] \times_{\texttt{LEX}} R\right) \quad \times_{\texttt{DIR}} \quad \sigma_S\left([\leqslant 2] \times_{\texttt{LEX}} S\right) \quad \times_{\texttt{LEX}} \quad T\right)\right)$$

where:

- The selection σ_R selects tuples with the criterion $(.1 = 1 \wedge .2 \neq 0) \vee (.1 = 2 \wedge .2 = 0)$
- The selection σ_S selects tuples with the criterion $(.1 = 2 \wedge .2 \neq 0) \vee (.1 = 1 \wedge .2 = 0)$
- The selection $\sigma_=$ selects tuples with the criterion $.1 = .5 \wedge .2 = .6 \wedge .3 = .7 \wedge .4 = .8$
- The projection Π projects on attribute 9.

We now explain, given the ab-bipartite po-relation $\Gamma_{\mathsf{ab}} = (ID, T, <)$, how we interpret the relation names R, S, T, and W with unordered relations. Let $U \sqcup V$ be the partition of ID into a-labeled and b-labeled elements, let $n := |U|$ and $m := |V|$, and write $U = (u_1, \ldots, u_n)$ and $V = (v_1, \ldots, v_m)$ following some arbitrary order. We define the input po-database D as follows:

- R is interpreted as an unordered po-relation containing tuples labeled $0, 1, \ldots, n$
- S is interpreted as an unordered po-relation containing tuples labeled $0, 1, \ldots, m$
- T is interpreted as an unordered po-relation containing the following tuples:
  - $\{\langle 1, i, 1, 0, \mathsf{a}\rangle \mid 1 \leqslant i \leqslant n\}$
  - $\{\langle 1, i, 2, j, \mathsf{c}\rangle \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m, \text{such that we have } u_i < v_j \text{ in } \Gamma_{\mathsf{ab}}\}$
  - $\{\langle 2, 0, 2, j, \mathsf{b}\rangle \mid 1 \leqslant j \leqslant m\}$
- W is interpreted as an unordered po-relation containing $i \times j$ tuples with label c.

The construction that we have described is clearly in PTIME.

Towards showing correctness, we first explain how query evaluation proceeds. First, $\sigma_R([\leqslant 2] \times_{\texttt{LEX}} R)$ creates a po-relation Γ'_R (illustrated in Figure 7) with a tuple $id_{1,i}$ labeled $\langle 1, i\rangle$ for $1 \leqslant i \leqslant n$ and a tuple $id_{2,0}$ labeled $\langle 2, 0\rangle$, with $id_{1,i} < id_{2,0}$ for all $1 \leqslant i \leqslant n$ and no other comparability pairs. Likewise, $\sigma_S([\leqslant 2] \times_{\texttt{LEX}} S)$ creates a po-relation Γ'_S (also illustrated in Figure 7) with a tuple $id_{1,0}$ labeled $\langle 1, 0\rangle$ and tuples $id_{2,j}$ labeled $\langle 2, j\rangle$ for $1 \leqslant j \leqslant m$, with $id_{1,0} < id_{2,j}$ for all $1 \leqslant j \leqslant m$ and no other comparability pairs.

We now do the $\times_{\texttt{DIR}}$ product of Γ'_R and Γ'_S, and write $\Gamma' := \Gamma'_R \times_{\texttt{DIR}} \Gamma'_S$: see Figure 8 for an illustration. Formally, Γ' has four kinds of identifiers:

- Tuples $id_{1,i,1,0}$ with label $\langle 1, i, 1, 0\rangle$ for $1 \leqslant i \leqslant n$. These are pairwise incomparable, because the $id_{1,i}$ were.
- Tuples $id_{2,0,2,j}$ with label $\langle 2, 0, 2, j\rangle$ for $1 \leqslant j \leqslant m$, which are also pairwise incomparable.
- Tuples $id_{1,i,2,j}$ with label $\langle 1, i, 2, j\rangle$ for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$, which are pairwise incomparable.
- One tuple $id_{2,0,1,0}$ with label $\langle 2, 0, 1, 0\rangle$.

Intuitively, the identifier $id_{1,i,1,0}$ will represent element u_i, the identifier $id_{2,0,2,j}$ will represent element v_j, the identifier $id_{1,i,2,j}$ will represent an edge between u_i and v_j (denoted $e_{i,j}$), and the identifier $id_{2,0,1,0}$ is not important and will be removed soon by the selection $\sigma_=$. Note that we have an element $e_{i,j}$ for all pairs of identifiers in $U \times V$, no matter whether they are comparable in Γ_{ab}.

As for the order relations across identifiers of different kinds, they are as follows:

- For $1 \leqslant i \leqslant n$, the identifier $id_{1,i,1,0}$ is less than the identifier $id_{2,0,1,0}$ and it is less than the identifiers $id_{1,i,2,j}$ and the identifiers $id_{2,0,2,j}$ for all $1 \leqslant j \leqslant m$.
- The identifier $id_{2,0,1,0}$ is less than the identifiers $id_{2,0,2,j}$ for all $1 \leqslant j \leqslant m$.
- For all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$, the identifier $id_{1,i,2,j}$ is less than the identifier $id_{2,0,2,j}$.

Forgetting about $id_{2,0,1,0}$ which is not important, the intuition is that we have $u_i < e_{i,j} < v_j$ for all i and j.

We have described $\Gamma' = \Gamma'_R \times_{\mathtt{DIR}} \Gamma'_S$, and we continue describing how query evaluation proceeds. The intuition now is that we wish to only keep the $e_{i,j}$ such that $u_i < v_j$ in Γ_{ab}. We cannot express this directly using a selection, because the selection criterion would depend on Γ_{ab} so it would not be fixed. Instead, we take the product of Γ' with the unordered po-relation T that specifies which identifiers we wish to keep, and then we perform the fixed selection $\sigma_=$. Specifically, we do the $\times_{\mathtt{LEX}}$ product of Γ' with T, followed by the selection $\sigma_=$, which intuitively replaces each element of Γ' by the contents of relation T, i.e., unordered identifiers, so the order relation in the result of the $\times_{\mathtt{LEX}}$ product is entirely defined by the first component, i.e., by Γ'. The selection $\sigma_=$ then keeps the $id_{1,i,2,j}$ such that $u_i < u_j$, and it also keeps the $id_{1,i,1,0}$ and the $id_{2,0,2,j}$; it discards the other $id_{1,i,2,j}$ as well as the unimportant identifier $id_{2,0,1,0}$. After the selection, we perform a projection Π to rename the identifiers using the last component of the tuple labels in T: identifiers that come from the $id_{1,i,1,0}$ are relabeled a, identifiers that come from the $id_{2,0,2,j}$ are relabeled b, and identifiers that come from the $id_{1,i,2,j}$ are relabeled c. Last, we do the union with W to add $n \times m$ unordered identifiers labeled c.

To summarize, the po-relation $Q(D)$ contains the following identifiers:

- $m \times n$ unordered identifiers labeled c, each of which is incomparable to all other identifiers.
- n identifiers corresponding to the $id_{1,i,1,0}$ in Γ' for $1 \leqslant i \leqslant n$, that are labeled a, and that are incomparable among themselves: we identify each $id_{1,i,1,0}$ to the identifier u_i in Γ_{ab}.
- m identifiers corresponding to the $id_{2,0,2,j}$ in Γ' for $1 \leqslant j \leqslant m$, that are labeled b, and that are incomparable among themselves: we identify each $id_{2,0,2,j}$ to the identifier v_j in Γ_{ab}.
- One identifier corresponding to $id_{1,i,2,j}$ in Γ' for each $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$ such that $u_i < v_j$ is a comparability pair in Γ_{ab}: we call each of them $e_{i,j}$ for brevity.

The comparability pairs across these identifiers are simply the following: $u_i < v_j$ for all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant n$, and $u_i < e_{i,j} < v_j$ for all i and j such that $e_{i,j}$ exists. In particular, note that the order between the u_i and v_j is *not* like in Γ_{ab}, because all u_i are less than

all v_j. We will work around this issue when defining our candidate possible world to read the comparability relation from the $e_{i,j}$.

To define the candidate possible world, consider now the integers $p, q \in \mathbb{N}$ that were given as input to the NP-hard problem of Proposition 94 along with Γ_{ab}. Let $0 \leqslant \pi \leqslant m \times n$ be the number of comparability pairs of Γ. Construct the totally ordered po-relation $L'_{p,q} := \mathsf{a}^p\mathsf{c}^{m\times n}\mathsf{a}^{u-p}\mathsf{b}^q\mathsf{c}^\pi\mathsf{b}^{v-q}$, which we will use as our candidate possible world. We claim that the POSS problem for $L_{p,q}$ and Γ_{ab} reduces to the same problem for $L'_{p,q}$ and $Q(D)$, which suffices to conclude the proof.

In one direction, assume that $L_{p,q} \in pw(\Gamma_{\mathsf{ab}})$, and consider a witnessing linear extension. We build a linear extension of $Q(D)$ achieving $L'_{p,q}$ as follows:

1. Enumerate the same a-labeled identifiers in $Q(D)$ as the ones in the witnessing linear extension of Γ_{ab} that achieves the factor a^p of $L_{p,q}$.
2. Enumerate all the $e_{i,j}$ that can be enumerated: there are at most $m \times n$ in total so we can enumerate all that are available at this point.
3. Enumerate some c-labeled identifiers from W afterwards if necessary, to enumerate $m \times n$ c-labeled identifiers in total.
4. Enumerate all remaining a-labeled identifiers.
5. Enumerate the same b-labeled identifiers in $Q(D)$ as the ones in the witnessing linear extension of Γ_{ab} that achieves the factor b^q of $L_{p,q}$. To see why these identifiers can be enumerated at this stage in $Q(D)$, assume by way of contradiction that we try to enumerate v_j in $Q(D)$ but that this violates an order constraint of $Q(D)$, i.e., v_j is greater than another identifier that has not been enumerated yet. As all a-labeled identifiers of $Q(D)$ have been enumerated in steps 1 and 4, the only identifiers that can block the b-labeled identifier v_j from being enumerated are the c-labeled identifiers $e_{i,j}$ that have not been enumerated at step 2. So there must be $1 \leqslant i \leqslant n$ such the identifier $e_{i,j}$ exists in $Q(D)$ and was not enumerated at step 2. Now, the only way for this to happen is if the a-labeled element u_i was not enumerated at step 1. However, the existence of $e_{i,j}$ in $Q(D)$ witnesses that $u_i < v_j$ in Γ_{ab}, and in the linear extension of Γ_{ab} we must have enumerated u_i before v_j. Hence, u_i was enumerated at step 1 and $e_{i,j}$ was enumerated at step 2 and v_j can now be enumerated, a contradiction.
6. Enumerate the remaining c-labeled identifiers and b-labeled identifiers arbitrarily, which is clearly possible as no comparability pairs between unenumerated elements remain.

In the converse direction, assume that $L'_{p,j} \in pw(Q(D))$, and consider a witnessing linear extension. We build a linear extension of Γ_{ab} achieving L_i by matching the factors a^p and b^q to the elements matched to these factors in $Q(D)$, and finishing by enumerating the remaining a-labeled and b-labeled elements in some arbitrary way. The only thing to show is that we do not violate the order constraints of Γ_{ab} while achieving the factors a^p and b^q. To show this, assume by way of contradiction that we try to enumerate some identifier v_j when achieving b^q but we have $u_i < v_j$ for some identifier u_i that was not enumerated when achieving a^p. In this case, the comparability pair $u_i < v_j$ of Γ_{ab} witnesses the existence of an element $e_{i,j}$ in $Q(D)$ such that $u_i < e_{i,j} < v_j$ in $Q(D)$. Now, as we did not enumerate u_i to achieve a^p in $Q(D)$, we cannot have enumerated $e_{i,j}$ when achieving $\mathsf{c}^{m\times n}$, hence $e_{i,j}$ witnesses that we cannot have enumerated v_j when achieving b^q in $Q(D)$, a contradiction. Hence, the order constraints of Γ_{ab} are respected.

This concludes the correctness argument, so we have shown the NP-hardness of POSS in our context, which concludes the proof of Theorem 92. ◀

References for the Appendix

ABDS17a A. Amarilli, M. L. Ba, D. Deutch, and P. Senellart. Possible and certain answers for queries over order-incomplete data. In *Proc. TIME*, 2017.

ABDS17b A. Amarilli, M. L. Ba, D. Deutch, and P. Senellart. Possible and certain answers for queries over order-incomplete data. *CoRR*, 1707.07222v1, 2017. Version 1.

ABDS18 A. Amarilli, M. L. Ba, D. Deutch, and P. Senellart. Computing possible and certain answers over order-incomplete data. *CoRR*, 1801.06396, 2018.

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.

Hir55 T. Hiraguchi. On the Dimension of Orders. *Sci. rep. Kanazawa Univ.*, 4(01), 1955.

Øre62 O. Øre. Partial order. In *Theory of Graphs*, chapter 10. AMS, 1962.

Pin97 J.-E. Pin. Syntactic semigroups. In *Handbook of Formal Languages*, chapter 10. Springer, 1997.

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

WH84 M. K. Warmuth and D. Haussler. On the complexity of iterated shuffle. *JCSS*, 28(3), 1984.