Provenance for Nondeterministic Order-Aware Queries

Antoine Amarilli
Télécom ParisTech; CNRS LTCI

M. Lamine Ba
Télécom ParisTech; CNRS LTCI

Daniel Deutch
Tel Aviv University

Pierre Senellart
Télécom ParisTech; CNRS LTCI

ABSTRACT

Data transformations that involve (partial) *ordering*, and consolidate data in presence of *uncertainty*, are common in the context of various applications. The complexity of such transformations, in addition to the possible presence of meta-data, call for *provenance support*. We introduce, for the first time, a framework that accounts for the conjunction of these needs. To this end, we enrich the positive relational algebra with order-aware operators, some of which are *non-deterministic*, accounting for uncertainty. We study the expressive power and the complexity of deciding possibility for the obtained language. We then equip the language with (semiring-based) *provenance tracking* and highlight the unique challenges in supporting provenance for the order-aware operations. We explain how to overcome these challenges, designing a new provenance structure and a provenance-aware semantics for our language. We show the usefulness of the construction, proving that it satisfies common desiderata for provenance tracking.

1. INTRODUCTION

Real world applications often involve transformations that involve some (partial) *ordering* in the data; that need to consolidate the data in presence of *uncertainty*; and that can greatly benefit from *provenance support* due to their complexity and dependency on meta-data. We define and study a framework that addresses, for the first time (to our knowledge), the combination of these three challenges. We first explain the need for such a framework and then highlight our main contributions in this respect.

Ordered Data and its Consolidation. Queries involving some form of (partial) ordering in the data, such as sorting (`ORDER BY` in SQL), queries using ordered timestamps, or top-k queries (`LIMIT`), play a fundamental role in many applications. When ordered datasets are *consolidated*, their management becomes intricate. For example, what should be the ordering of the union of ordered relations that may or may not be disjoint? If they are not disjoint, how to solve conflicts in the ordering? Assuming a global, total order on all tuples is unrealistic in practice: perhaps the relations in question were ordered by different incomparable attributes; or the attribute used to order them was projected out; or, if the relations come from different sources, each may be ordered following local, unsynchronized timestamps. Such challenges are present also in the context of *rank aggregation* [29] when the individual aggregation functions are unknown and lists must be merged and ordered in a way compatible with the individual orderings; or for *scheduling of workflows*, with constraints on tasks order and possible synchronization points. In all of these cases there is an inherent *uncertainty* in the transformations. As explained below, we take the operational approach of dealing with this uncertainty via *non-determinism*.

Consider for example a *sensor network* where each sensor issues observations on events happening within its range. We assume that information about events observed by a given sensor is saved in a relation and are ordered by timestamps. Observations of the different sensors need to be consolidated, to provide a complete picture of events and allow for their analysis. However, we may not trust the relative ordering of observations across sensors, as global clock synchronization is a tricky matter [30]; or maybe we can trust the relative ordering between sensors but only once some synchronization point has been reached (e.g. an event that is known to be common has been reported).

A Need for Provenance Tracking. Importantly, *meta-data* may affect the transformation and consolidation of data. Continuing with our sensors example, each observation of each sensor may be associated with a different level of credibility (trust), depending e.g. on the sensor quality; some observations may be associated with different *access control privileges*, so that the result of data consolidation must not show them to unauthorized users; etc. Also, due to the complicated nature of such transformations, users may wish to explore multiple *scenarios*, both with respect to presence or absence of input data ("how would the result change if we omit a particular observation?"), or with respect to the multiple possible ways for integration. *Provenance management* has proven highly successful for such applications, in the context of different kinds of transformations (e.g. [5, 26, 31, 17, 11, 20]), and so we identify the *need for provenance tracking* in the context of order-aware transformations.

To our knowledge, no previously proposed framework can be used for our needs. For instance, standard SQL is unsuitable as "ordering of the rows of the table specified by the query expression is guaranteed only for the query expression that immediately contains the `ORDER BY` clause" [23], which means ordering is not preserved except at top-level. Existing works on querying in presence of order typically do not admit a nondeterministic semantics: they either assume a *total* order on the data elements [22, 32], forbid conflicts [27], or choose one possible merge based on heuristics or rules [4, 25]. Other works such as [21] propose to capture results via partially-ordered sets (with multiplicities), and we discuss the unsuitability of this approach to our needs in Section 6. In

addition, to our knowledge, provenance management of the aforementioned flavor has not been studied in the presence of order, or for non-deterministic query languages.

We therefore propose in this paper a new non-deterministic order-aware query language for relational data, study its properties in terms of expressivity and complexity, and equip it with a specifically designed provenance model. We next provide an overview of our main contributions.

A Nondeterministic Order-Aware Query Language (Section 2). We focus on a relational setting, and capture order by introducing a dedicated attribute **Ord** whose values are natural numbers reflecting the relative order of a tuple. We then define four new operations on relations, that pertain to order: (i) a unary deterministic "constructor" that turns an unordered relation into an ordered one (by introducing the **Ord** attribute carrying the location of tuples in the order), based on some given order specification, and (ii) three binary operations performing "order-aware union" on ordered relations: Concat to deterministically concatenate tuples of the first relation to those of the second (keeping their relative order); Shuffle to *nondeterministically* choose an order over the union of tuples that is consistent with the orders at the two input relations; and Sync, which is similar to Shuffle (and again nondeterministic) but uses matching tuples as synchronization points.

We then define the OWALG language (for "order aware algebra") as the language including these operators in addition to the standard SPJ operators (regular union being expressible with the new operators). With the exception of some restrictions imposed on joins, the SPJ operators may treat the **Ord** attribute as any other attribute and interact with the non-deterministic operators by being applied in every possibly generated world[1]. We focus in this paper on *set semantics* for the language, allowing no duplicates (tuples that have the same values in all attributes, including **Ord**). This is due to some inherent difficulties in the interaction of duplicates, order and provenance, that we point out.

Expressiveness and Complexity (Sections 3 and 4). Before we study provenance for OWALG, we formalize its connection to partial orders [28] (saying that an OWALG query expresses a partial order P if its possible worlds are the total orders consistent with P) and use this to study the expressiveness of OWALG. We show that OWALG *can represent any partial order*, and conversely that any OWALG query of a certain class essentially represents a partial order. We also show that the fragment obtained by disallowing the Sync operator expresses exactly the *series-parallel* [28] class of partial orders. Building on this connection with order theory, we determine the complexity of the *possibility problem* (identifying if a given instance is a possible world of an OWALG query), showing that it is NP-complete in data complexity, and identifying a tractable case.

Provenance (Section 5). We then present a provenance model for the language. Frameworks based on algebraic structures have proven successful in the context of the positive relational algbera [20], queries with difference [18], queries on (unordered) XML [15], SPARQL queries [17], and others, for goals such as the incorporation of meta-data, explanation of results and explorations of scenarios (exemplified above). For this reason, we develop an appropriate algebraic structure for provenance tracking for OWALG and a corresponding provenance-aware semantics for the language.

To do so, we need to address three main challenges. First, to our knowledge, semiring-based provenance has never been studied in the context of *ordered domains*, as existing work focuses on unordered relations [20], unordered XML [15], etc. Second, our treatment of order involves non-deterministic queries, while previous provenance constructions, though they allow non-determinism in the input data (as we do), assume *deterministic queries*. Third, to capture provenance, the tuple annotations need to be *combined* with the ordered values in a well-principled algebraic manner (the latter in a similar way to the case of aggregate queries [2]).

This interplay of order, non-determinism and annotations leads to novel challenges. To this end, we develop a suitable algebraic structure, and show that it can be used to support provenance. We justify the usefulness of our provenance framework by proving that it satisfies the standard desiderata of [2] for such algebraic provenance frameworks, with one notable exception caused by inherent limitations in the treatment of *bag semantics*.

We discuss related work in Section 6 and conclude with directions for future work in Section 7. Due to space restrictions, all complete proofs are deferred to the appendix.

2. ORDER-AWARE ALGEBRA

We assume that the reader is familiar with the positive relational algebra [1]. We start with the select-project-join (SPJ) fragment for which we assume set semantics, and we enrich it with a suite of new order-aware operators (equivalent to the standard union operator in absence of order) whose syntax and semantics is next introduced. To simplify the presentation we assume for now that (1) SPJ operators are never applied to the result of an order-aware operator; (2) no operators are applied to the result of non-deterministic operators. This is just to ease presentation, and both restrictions are lifted in Section 2.5 we alleviate both assumptions.

Notations. Given a relation R we use $\text{attr}(R)$ to denote its set of attributes. For a tuple $r \in R$ and attribute $a \in \text{attr}(R)$, we use $r.a$ to denote the value of attribute a in r. We will frequently project out attributes from a relation R or a tuple $r \in R$, i.e., $\Pi_{\text{attr}(R)\setminus A}(R)$ or $\Pi_{\text{attr}(R)\setminus A}(t)$ to project out A, which we write as $\Pi_{-A}(R)$ and $\Pi_{-A}(t)$.

Order. We consider one particular attribute, denoted **Ord**, that we will use to carry information about the ordering of data. Its domain is the natural numbers. We then say that a relation is *ordered* if it has an attribute **Ord** whose domain is totally ordered and no two tuples of R have the same **Ord** value (ties will be considered in Section 2.5). As we only care about the relative order of tuples rather then the exact **Ord** values, we say that **Ord** is defined up to order-preserving isomorphism, namely, two relations R and R' are equivalent (and we simply write $R = R'$) if there is an order-preserving

[1]This is reminiscent of the approach in e.g. [19] to deal with uncertainty in a different context of key violations.

ID	Read	Loc	Time	**Ord**
R0	Light	1	3:30:02	0
R1	Door	1	3:30:04	2
R2	Light	3	3:30:03	1

(1.a) Relation $R = \text{Rank}_{\mathsf{Time},<}(R_1)$

ID	Read	Loc	Time	**Ord**
T0	Beep	1	3:29	0
T1	Door	1	3:31	1

(1.b) Relation T

ID	Read	**Ord**
T0	Beep	0
R0	Light	1
R1	Door	2
R2	Light	3
T1	Door	4

(1.c) Result of q_{concat}

Read	**Ord**
Beep	0
Light	1
Door	2
Light	3
Door	4

(1.d) Poss. for q_s

Read	**Ord**
Light	0
Door	1
Light	2
Door	3
Beep	4

(1.e) Impos. for q_s

isomorphism f mapping the domain of $R.\mathbf{Ord}$ to that of $R'.\mathbf{Ord}$ so that $f(R) = R'$.

2.1 Rank

The first operator is a "constructor" for ordered relations, that turns an unordered relation into an ordered one by leveraging an input order relation over preexisting attributes.

DEFINITION 2.1. *Let R be a relation and $A \in \text{attr}(R)$ such that the domain of values in $\Pi_A(R)$ is totally ordered by $<$ and $\Pi_A(R)$ includes no duplicates*[2]*. Then $\text{Rank}_{A,<}(R)$ is defined as a relation S satisfying:*

- $\text{attr}(S) = \text{attr}(R) \cup \{\mathbf{Ord}\}$
- $\Pi_{-\mathbf{Ord}}(S) = \Pi_{-\mathbf{Ord}}(R)$
- *For each $s \in S$, $s.\mathbf{Ord} = |\{s' \in S \mid s'.A < s.A\}|$*

Observe that the definition also allows the application of Rank to an ordered relation for ranking it anew, in which case the existing **Ord** attribute of the relation is discarded.

EXAMPLE 2.2. *Consider relation R from Table 1.a without column* **Ord***, and call this R_1. Table 1.a represents $R = \text{Rank}_{\mathsf{Time},<}(R_1)$ with $<$ the usual order relation on* Time*.*

Even though we have defined equivalence between ordered relation up to order-preserving isomorphism, it will sometimes be convenient to assume that the values in the **Ord** attribute are consecutive Interestingly, this may easily be enforced by re-applying Rank:

PROPOSITION 2.3. *Given an ordered relation R, the relation $\text{Rank}_{\mathbf{Ord},<}(R)$ (where the "<" has its standard semantics on natural numbers) is equivalent to R but is such that the values of* **Ord** *are consecutive integers from 0 to $|R|-1$.*

So, up to an additional application of Rank in every subexpression, we can assume that the domain of **Ord** is always a sequence of consecutive integers.

2.2 Concat

The Concat operator takes the union of two relations R, R' and orders them by placing all elements of R (in order) before all elements of R' (in order). If a tuple (maybe up to its **Ord** attribute) appears in both R and R', there will be two counterparts of this tuple, with different **Ord** values, in the output of Concat.

DEFINITION 2.4. *Given two relations R, R' such that $\mathbf{Ord} \in \text{attr}(R) = \text{attr}(R')$, $\text{Concat}(R, R')$ is the relation S with $\text{attr}(S) = \text{attr}(R) = \text{attr}(R')$ such that there is an isomorphism h between the* bag union[3] *$R \cup R'$ and S satisfying:*

- *For all $s \in S$, $\Pi_{-\mathbf{Ord}}(s) = \Pi_{-\mathbf{Ord}}(h^{-1}(s))$.*
- *If $s = h(r)$ such that $r \in R$ then $s.\mathbf{Ord} = |\{r_1 \in R \mid r_1.\mathbf{Ord} < r.\mathbf{Ord}\}|$.*
- *If $s = h(r')$ such that $r' \in R'$, then $s.\mathbf{Ord} = |\{r'_1 \in R' \mid r'_1.\mathbf{Ord} < r'.\mathbf{Ord}\}| + |R|$.*

EXAMPLE 2.5. *Consider the relations R and T of Tables 1.a and 1.b. The granularity of their* Time *column differs, but we can consolidate them with* Concat *to specify that the tuples of T with timestamp $\leqslant$3:30 (resp., >3:30) should precede (resp., follow) all the tuples of R.*

$$q_{concat} = \text{Concat}(\text{Concat}(q_1, R), q_2) \quad \text{where} \begin{cases} q_1 = \text{Rank}_{\mathsf{Time},<}(\sigma_{T.\mathsf{Time} \leqslant 3:30}(T)) \\ q_2 = \text{Rank}_{\mathsf{Time},<}(\sigma_{T.\mathsf{Time} > 3:30}(T)). \end{cases}$$

The query result (omitting for brevity the Loc and Time attributes) is given as Table 1.c. Observe that the definition of Concat *does not require unique IDs, so without the* ID *column we would simply have multiple tuples in the result distinguished only by their* **Ord** *value (but since* **Ord** *is one of the attributes, this is still set semantics).*

2.3 Shuffle

The Concat operator sets a particular order between tuples of the relations on which it is applied; however, there is often uncertainty in the relative order of tuples. We thus introduce the Shuffle operator, which unions elements of two ordered relations and *nondeterministically* chooses a new total order over the tuples, respecting the orders of the original relations.

For an ordered relation R and tuples $r_1, r_2 \in R$, we say that r_1 *precedes* r_2 (and r_2 *follows* r_1) if $r_1.\mathbf{Ord} < r_2.\mathbf{Ord}$.

DEFINITION 2.6. *Given two relations R, R' such that $\mathbf{Ord} \in \text{attr}(R) = \text{attr}(R')$, $\text{Shuffle}(R, R')$* nondeterministically chooses *a relation S, with $\text{attr}(S) = \text{attr}(R) = \text{attr}(R')$, with an isomorphism h between $R \cup R'$ (bag union) and S, satisfying:*

- *For all $s \in S$, $\Pi_{-\mathbf{Ord}}(s) = \Pi_{-\mathbf{Ord}}(h^{-1}(s))$.*
- *If $s = h(r)$ such that $r \in R$, s precedes every $s_1 = h(r_1)$ such that $r_1 \in R$ precedes r.*
- *If $s = h(r')$ such that $r' \in R'$, s precedes every $s_1 = h(r'_1)$ such that $r'_1 \in R'$ precedes r'.*

We say that the set of relations S satisfying the above is the set of possible worlds *of* $\text{Shuffle}(R, R')$.

Observe that $\text{Concat}(R, R')$ is always one of the possible worlds of $\text{Shuffle}(R, R')$.

EXAMPLE 2.7. *Reconsider R and T from Tables 1.a and 1.b. Further consider the query:*

$$q_s = \text{Shuffle}(\Pi_{\mathsf{Read},\mathbf{Ord}}(R), \Pi_{\mathsf{Read},\mathbf{Ord}}(T)).$$

[2]This assumption is relaxed in Section 2.5.

[3]This means in particular that if a tuple appears in both R and R', each of its occurrences is mapped to a different tuple in S

Intuitively, this query defines a set of possible worlds, corresponding to the possible consistent orders interleaving the tuples of R and T. Table 1.d is a possible world of the query, but Table 1.e is not*, intuitively because relation T requires a* Door *event to follow the* Beep *event.*

2.4 Sync

Intuitively, in Example 2.7, we assume that the events seen by the different sensors are distinct: if a "Door" event is reported by two sensors, Shuffle considers that they were two different such events (which may be reasonable if e.g. values in the Loc attribute of our example refer to large areas). Alternatively, if, e.g., the sensors are monitoring the same *room*, one may wish to "synchronize" the two event occurrences reported in both relations.

This motivates the introduction of the Sync operator. Its definition differs from that of Shuffle for the tuples which (up to the **Ord** attribute) occur in both relations. While Shuffle kept both tuples, Sync retains only one, in a way that is consistent with both input relations.

In order to have no ambiguities when matching event occurrences, we assume as a precondition to the use of Sync that each of its individual input relations R, R' does not contain two *duplicate* tuples $t \neq t'$ that are the same up to the **Ord** relation (i.e., $\Pi_{-\mathbf{Ord}}(t) = \Pi_{-\mathbf{Ord}}(t')$). Note that, since we use set semantics, this is in particular true whenever R and R' are the result of an application of the Rank operator.

Before defining Sync, we note a subtelty in the way it deals with contradictory information from both relations. We first define the notion of *conflict*:

DEFINITION 2.8. *We say that t and t'* conflict *in R and R' if $\Pi_{-\mathbf{Ord}}(t), \Pi_{-\mathbf{Ord}}(t') \in \Pi_{-\mathbf{Ord}}(R) \cap \Pi_{-\mathbf{Ord}}(R')$ and we have: $R.\mathbf{Ord}(t) < R.\mathbf{Ord}(t')$ but $R.\mathbf{Ord}(t') < R.\mathbf{Ord}(t)$.*

When possible, Sync will make a non-deterministic choice of "solving" the conflict for every pair of tuples. As we shall show (Example 2.11) this may not be possible, and then we allow Sync to *fail*.

We next introduce the semantics, then exemplify a case where it succeeds and one where it must fail (i.e., fails in every possible world).

DEFINITION 2.9. *Given two relations R, R' such that $\mathbf{Ord} \in \mathrm{attr}(R) = \mathrm{attr}(R')$ and R, R' contain no duplicates up to* **Ord**, $\mathrm{Sync}(R, R')$ *nondeterministically chooses a relation S, with $\mathrm{attr}(S) = \mathrm{attr}(R) = \mathrm{attr}(R')$, such that either $S = \emptyset$ or S satisfies the following:*

- $\Pi_{-\mathbf{Ord}}(S) = \Pi_{-\mathbf{Ord}}(R) \cup \Pi_{-\mathbf{Ord}}(R')$ *(set union).*
- *For all t_1, t_2 in S, if t_1 precedes t_2 in S, if there are tuples $t_1' \simeq t_1$, $t_2' \simeq t_2$ in one input table such that t_2' precedes t_1', there are tuples $t_1'' \simeq t_1$, $t_2'' \simeq t_2$ in the other input table such that t_1'' precedes t_2''.*

where we write $t \simeq t'$ for $\Pi_{-\mathbf{Ord}}(t) = \Pi_{-\mathbf{Ord}}(t')$.

Note that if the new ordering violates one of the two existing orderings, it is necessarily because these orderings disagree. Conversely, whenever input relations agree on the ordering of two tuples, the result of Sync will follow the consensus. The empty relation is included for technical reasons, as a *default* outcome: there may be *no relation S* satisfying the constraints, or the user may find no possible merging option acceptable, but we want to guarantee that there is always at least one possible choice for the operator. In this latter case, we say that Sync *fails*, and otherwise that it *succeeds*.

EXAMPLE 2.10. *Reconsider R and T from Tables 1.a and 1.b, and define $R' = \sigma_{\mathsf{Loc}=1}(R)$. Assume that the* Time *and* IDs *are local to each sensor (i.e., to each relation), but that the two events of type* Door *occurring at location 1 are actually the same event witnessed by both sensors.*

Consider therefore the relations $R'' = \Pi_{-\{\mathsf{ID},\mathsf{Time}\}}(R')$ and $T'' = \Pi_{-\{\mathsf{ID},\mathsf{Time}\}}(T)$. The Shuffle *operator is not suitable for our purpose, because its possible worlds will contain two occurrences of the* Door *event. In contrast, the* Sync *operator merges those two occurrences and uses them as a synchronization point, so that the* Light *and* Beep *events must take place before the* Door *event (but their relative order is still uncertain as each was witnessed by only one sensor). The possible worlds of* $\mathrm{Sync}(R'', T'')$ *are thus:*

Read	Loc	**Ord**
Light	1	0
Beep	1	1
Door	1	2

Read	Loc	**Ord**
Beep	1	0
Light	1	1
Door	1	2

The following example shows a case where Sync must fail.

EXAMPLE 2.11. *Consider the two relations U and U':*

Read	**Ord**
Beep	0
Door	1
Light	2

Read	**Ord**
Light	0
Heat	1
Beep	2

As constraints on the possible worlds of Sync *we have (from U)* Beep *must precede* Door *and* Door *must precede* Light*, so* Beep *must precede* Light*, but symmetrically (from U')* Light *must precede* Beep*, so* $\mathrm{Sync}(U, U')$ *must fail: there is no S satisfying the constraints. Intuitively, any solution of this conflict will either allow no correct location of* Heat *or no correct placement of* Door.

The following propositions present natural cases in which Sync will never be forced to fail.

PROPOSITION 2.12. *If no t and t' conflict in R and R' then* $\mathrm{Sync}(R, R')$ *may succeed unless R and R' are both empty.*

PROPOSITION 2.13. *If R and R' have the same elements irrespectively of order, i.e., $\Pi_{-\mathbf{Ord}}(R) = \Pi_{-\mathbf{Ord}}(R')$, then* $\mathrm{Sync}(R, R')$ *may succeed unless R and R' are both empty.*

2.5 OWALG

We define OWALG as the algebra including the relational selection, projection, and join (SPJ) operators under the usual set semantics, as well as the four order-aware operations just introduced. As mentioned above, when applied to unordered relations, the semantics for SPJ is the standard set semantics with duplicate elimination. We next lift the two assumptions made earlier, i.e., (1) define the result of applying SPJ operators on ordered relations, and (2) define the result

of applying (SPJ or order-aware) operators on the result of non-deterministic sub-queries.

For (1), to the exception of the subtlety of the next paragraph, our definition of ordered relations allows SPJ operations to treat them as any other relation, in particular treating **Ord** as any other attribute. This is useful to, e.g., select the i-th element of the order with a selection on **Ord** (assuming Rank is used to keep the **Ord** values consecutive, as explained above). Note that selection may lead to non-consecutive **Ord** values (but this may be fixed by applying Rank), and projecting out the **Ord** attribute will lead to an unordered relation.

The only subtlety is then in the Join operation. First, joining an ordered relation with an unordered one yields an ordered relation, possibly with multiple tuples with the same **Ord** values (in addition to possible gaps); hence, when applying the Rank operator to renumber **Ord** to contiguous values, these ties need to be broken, which is performed *nondeterministically* following the semantics for tied values which we will define shortly. Second, we do not allow joining two ordered relations, as we would not know in this case which of the input orders should be preserved: instead, one should first project out the **Ord** attribute on one input so that we are back to the previous case.

For (2), we use the intuitive approach of, e.g., [19]: whenever a sub-query generates a set of possible worlds, the rest of the query is applied on every one of these worlds. Note that if the next operator to be applied is again non-deterministic, this may lead to the generation of multiple possible worlds for each such world, etc. The set of possible worlds of the query is then defined as the union of all possible worlds obtainable in such a way (where no operations remain to be applied). We note a subtlety with respect to Sync: now, when we say that an application of Sync *fails*, we really mean that it fails for all possible worlds of the sub-queries provided as input.

We have therefore defined a semantics for OWALG.

Ties. We have assumed that there are no ties in the attributes to which Rank is applied. This does not restrict the expressive power (up to adding the empty relation as a possible world):

PROPOSITION 2.14. *Consider a schema* $\mathrm{attr}(R)$ *and* $A \in \mathrm{attr}(R)$ *such that there are duplicate values for A in R, and let* $<$ *be a total order on the domain of A. There is an* OWALG *query Q such that, for every relation R with schema* $\mathrm{attr}(R)$*, the possible worlds of* $Q(R)$ *are exactly the possible ordered relations obtained by ordering the tuples of R with* $<$ *based on their value on A (in addition to the empty relation), for all possible ways of breaking ties.*

PROOF SKETCH. We find two total orders over the tuples of relation R which extend $<$ in one direction and in the reverse direction, rank R according to either, and Sync the result, which can succeed by Proposition 2.13. □

Bag Semantics. For now, we have only looked at *set semantics*. We could also define a bag semantics for OWALG in the expected way, associating multiplicities with tuples and offsetting the **Ord** values according to them. However, the presence of order makes multiplicity a lot less attractive, as queries may force the explicit repetition of tuples, resulting in an exponential blowup of the representation

EXAMPLE 2.15. *Consider* $R = \{(a,n)\}$ *and* $R' = \{(b,n)\}$ *where n stands for multiplicity. One of the possible worlds of* $Q = \mathrm{Shuffle}(\mathrm{Rank}_{A,<}(R), \mathrm{Rank}_{A,<}(R'))$ *is the ordered relation consisting of a sequence of* $2n$ *tuples which are alternatively a and b. Because of the need to represent this alternation, and thus the order between the individual copies of the a and b tuple, the result of this query must be represented as* $2n$ *tuples with multiplicity* 1.

For this reason, and because of related inherent limitations in managing provenance with bags (section 5.5), we focus on set semantics in the present paper.

3. EXPRESSIVE POWER

We now consider the expressive power of OWALG. In particular, the nondeterministic order-aware operators Shuffle and Sync seem to enforce a partial order over the elements of their possible worlds. In this section, we formalize this intuition and characterize which orders may be thus represented.

Preliminaries. We recall definitions related to (labeled) partially ordered sets. See, e.g., [7] for details.

DEFINITION 3.1. *A* poset*, or partially ordered set, is a pair* $P = (V, <)$ *where* $<$ *is an irreflexive and transitive binary relation over* V*. An order is* total *if every pair of elements* $x, y \in V$ *is* comparable *(i.e.,* $x < y$ *or* $y < x$ *holds).*

The domain *of* $P = (V, <)$ *is* $\mathrm{dom}(P) = V$*.*

An extension *of a partial order* $P = (V, <)$ *is an order* $(V, <')$ *such that whenever* $x < y$ *for* $x, y \in V$ *then* $x <' y$*. A* linear extension *is an extension which is a total order.*

A labeled poset *is a structure* $(V, \Sigma, \mu, <)$ *where* $\mu : V \mapsto \Sigma$ *is a mapping from the vertices set* V *to the label set* Σ*, and* $<$ *is a partial order over* V*. The notions of total orders and extension are straightforwardly extended to labeled posets.*

DEFINITION 3.2. *The class of* series-parallel posets *is the class including all single-element orders and closed under the* series *and* parallel *composition operations. Given two series-parallel posets P and Q with disjoint domains the* series *composition of P and Q is the poset on* $\mathrm{dom}(P) \sqcup \mathrm{dom}(Q)$ *whose restriction to* $\mathrm{dom}(P)$ *and* $\mathrm{dom}(Q)$ *matches P and Q, and such that* $p < q$ *for any* $p \in \mathrm{dom}(P)$ *and* $q \in \mathrm{dom}(Q)$*. The* parallel *composition of P and Q is the poset on* $P \sqcup Q$ *whose restriction to* $\mathrm{dom}(P)$ *and* $\mathrm{dom}(Q)$ *matches P and Q, and such that no* $p \in \mathrm{dom}(P)$ *and* $q \in \mathrm{dom}(Q)$ *are comparable.*

A series-parallel labeled poset is a labeled poset whose underlying partial order is series-parallel.

EXAMPLE 3.3. *All total orders are series-parallel. In contrast, the "N-shaped" poset [21] with elements* $\{a, b, c, d\}$*, defined by* $a < b$*,* $a < c$*, and* $d < c$*, is not series-parallel.*

We next define how a query may capture a labeled poset:

DEFINITION 3.4. *For any labeled poset* $P = (V, \Sigma, \mu, <)$*, database D and* OWALG *query Q, if* $Q(D)$ *is an ordered relation, we say* $Q(D)$ represents P *if* $\Sigma = \prod_{A \in \mathrm{attr}(Q(D)) \setminus \textbf{Ord}} \mathrm{dom}(A)$ *(the product of the domains of the attributes of* $Q(D)$*) and the possible worlds of* $Q(D)$ *are exactly the linear extensions*

of P seen as an ordered relation in the straightforward way: for each $x \in \mathrm{dom}(P)$, the relation contains the tuple $(\mu(x), i)$ where the **Ord** *value i is the position of x in the linear extension.*

EXAMPLE 3.5. *Consider the labeled poset P defined over $\{a, a', b\}$ by the order relations $a < b$, $a' < b$ and the labeling function $\mu(a) = (Light, 1)$, $\mu(b) = (Door, 1)$, and $\mu(c) = (Beep, 1)$, so that $\Sigma = \{\mu(a), \mu(b), \mu(c)\}$. The* Sync *query from Example 2.10 represents the partial order P.*

Without Sync. We first consider the fragment of OWALG where the Sync operation is disallowed. We show that this fragment captures all series-parallel labeled posets, and (under a reasonable restriction) captures exactly these posets (along with the empty poset).

THEOREM 3.6. *For any product Σ of domains and series-parallel labeled poset P on Σ, there exists an* OWALG *query Q without* Sync *and a database D, both of size polynomial in P, such that $Q(D)$ represents P.*

Conversely, for any OWALG *query Q without* Sync *and database D, if $Q(D)$ is ordered and Q does not include a selection on the* **Ord** *attribute, there exists a series-parallel or empty labeled poset P of size polynomial in Q and D such that $Q(D)$ represents P.*

Interestingly, when a selection on the **Ord** attribute is allowed, the result may not be representable by a poset anymore, as the following example illustrates:

EXAMPLE 3.7. *Consider relations R and T from Tables 1.a and 1.b. Consider the query*

$$\Pi_{\mathsf{Read},\mathbf{Ord}}(\sigma_{\mathbf{Ord}=0}(\mathrm{Shuffle}(R,T)))$$

It has two possible worlds: a relation containing only a Light tuple (at position 0) and a relation containing only a Beep tuple (also at position 0). This is not representable by a poset.

Complete Language. When allowing Sync, for the expressiveness result we will need to forbid possible worlds in which Sync operators fail:

DEFINITION 3.8. *For an* OWALG *query Q and a database D, the possible worlds of $Q(D)$* up to failure *are the possible worlds of $Q(D)$ obtained by some sequence of nondeterministic choices in which no* Sync *operator fails. We say that $Q(D)$ is* completely failed *if it has no possible world up to failure.*

For the next result, we say that an OWALG query represents a poset up to failure, if its possible worlds up to failure correspond to linear extensions of the poset. Under this revised definition, OWALG *can capture every poset*, and conversely, if an OWALG query obeys the same restriction on selection as in the previous paragraph then it represents a poset unless it is completely failed.

THEOREM 3.9. *For any product Σ of domains and labeled poset P on Σ, there exist an* OWALG *query Q and database D, both of size polynomial in P, such that $Q(D)$ represents P up to failure.*

Conversely, for any OWALG *query Q and database D, if $Q(D)$ is ordered, Q does not include a selection on* **Ord***, and Q is not completely failed, then there exists a labeled poset P of size polynomial in Q and D such that $Q(D)$ represents P up to failure.*

4. COMPLEXITY OF POSSIBILITY

As OWALG allows nondeterministic operations with a possibly exponential number of possible worlds that cannot be all materialized, a natural way to characterize the complexity of query evaluation is the *possibility problem*: decide, given an ordered relation, if it is a possible query result, i.e., if it obeys the order constraints of the query and input database.

EXAMPLE 4.1. *Refer back to $S = \mathrm{Sync}(R'', T'')$ from Example 2.10. Assume that a third sensor in Location 1 is supposed to have recorded reliably all of the events which occurred, producing some table S'. We may want to know if the output of this sensor is consistent with that of the others, by checking if S' is indeed a possible world of S. While in the example there are only two possible worlds, in general an approach based on enumeration would be intractable.*

We formally define the possibility problem as follows:

DEFINITION 4.2. *The POSS problem is to decide, given a* OWALG *query Q, a database D and a relation I as input, whether I is a possible world of $Q(D)$.*

We first show that, in the general case, the POSS problem is NP-complete.

THEOREM 4.3. *POSS is NP-complete (in data complexity), even for the query $\Pi_A(\mathrm{Sync}(R, R'))$ and input relations R, R' such that $\Pi_{-\mathbf{Ord}}(R) = \Pi_{-\mathbf{Ord}}(R')$.*

Tractable Case. We now show a class of tractable cases for which POSS has PTIME complexity.

DEFINITION 4.4. *We say that a query Q is k-*tractable *w.r.t. a database D if Q is of the form $\Pi_A(\mathrm{Sync}(E_1, \cdots, \mathrm{Sync}(E_{k-1}, E_k)\cdots))$ where $\mathbf{Ord} \in A$ and $E_1, \ldots E_n$ are deterministic subqueries (i.e., do not involve* Sync *or* Shuffle*), and there is no conflict between any $E_i(D)$ and $E_j(D)$.*

EXAMPLE 4.5. *The* Sync *query of Example 2.10 is* 2*-tractable with respect to the example database, with $A = \{\mathsf{Read}, \mathsf{Loc}, \mathbf{Ord}\}$.*

Intuitively, the only nondeterminism in *k*-tractable cases is how the elements belonging to only one input relation should be interleaved, and the main difficulty when solving POSS is to match the projection of the candidate world with the possible worlds before projection.

Note that, even for 2-tractable cases, for an input database of size *n*, there can be $\binom{n}{n/2}$ possible worlds, so we cannot enumerate them. Despite this, we now show tractability for *k*-tractable cases (for fixed *k*):

THEOREM 4.6. *For any fixed k, the POSS problem restricted to inputs where the query is k-tractable with respect to the database, is in PTIME.*

PROOF SKETCH. We evaluate subexpressions $E_1, \ldots, E_n$ and present a dynamic algorithm to decide, given the result of these queries, whether some possible world of the shuffle has a projection which realizes the desired instance. The running time of the algorithm is polynomial in the database size with the exponent depending on k. □

We show in the appendix (via a reduction from MIN-SAT) that the exponential blowup of our algorithm w.r.t. k is unavoidable, unless $P = NP$, even if the query is very simple:

THEOREM 4.7. *POSS is NP-hard w.r.t. k for k-tractable inputs, even when every E_i is the identity query.*

5. PROVENANCE FOR OWALG

We consider in this section provenance tracking for OWALG. We start by recalling the standard approach of using semirings as annotations, as well as some natural desiderata for provenance tracking. While these desiderata are satisfied by the framework of [20] for the positive relational algebra, we highlight the novel challenges due to the incorporation of order-aware operators which require the development of an extended algebraic structure. We then introduce such a structure and use it to define provenance propagation for OWALG.

5.1 Algebraic Background

We review in this subsection the basic algebraic structures previously used in the context of provenance in [2, 20]. They will serve as a starting point for our construction.

A *commutative monoid* is an algebraic structure $(M, +_M, 0_M)$ where $+_M$ is an associative and commutative binary operation and 0_M is an identity for $+_M$. A *monoid homomorphism* is a mapping $h : M \to M'$ where M, M' are monoids, and $h(0_M) = 0_{M'}$, $h(a +_M b) = h(a) +_{M'} h(b)$. A *commutative semiring* is a structure $(K, +_K, \cdot_K, 0_K, 1_K)$ where $(K, +_K, 0_K)$ and $(K, \cdot_K, 1_K)$ are commutative monoids, $\cdot_K$ is distributive over $+_K$, and $a \cdot_K 0_K = 0 \cdot_K a = 0_K$. A *semiring homomorphism* is a mapping $h : K \to K'$ where K, K' are semirings, and $h(0_K) = 0_{K'}$, $h(1_K) = 1_{K'}$, $h(a +_K b) = h(a) +_{K'} h(b)$, $h(a \cdot_K b) = h(a) \cdot_{K'} h(b)$. Whenever we say semiring (monoid) in the sequel, we mean a commutative one.

Examples [20] include the *Boolean* semiring $(\mathbb{B}, \vee, \wedge, \bot, \top)$, the semiring $(\mathbb{N}[X], +, \cdot, 0, 1)$ of *polynomials* over a set of tokens X, the semiring of *natural numbers* $(\mathbb{N}, +, \cdot, 0, 1)$, and the *security* semiring $(\mathbb{S}, \min, \max, 0_{\mathbb{S}}, 1_{\mathbb{S}})$ where $\mathbb{S}$ is the ordered set, $1_{\mathbb{S}} < \mathsf{C} < \mathsf{S} < \mathsf{T} < 0_{\mathbb{S}}$ with the following respective interpretations: Public ("always available"); Confidential; Secret; Top secret; Never available.

Let K be a commutative semiring and M be a commutative monoid. We now define a slight variation of their *tensor product*, introduced in [2] for the support of aggregates: we intuitively "pair" elements of K with those of M, then impose desired axioms. We start by denoting a pair of elements $\langle k \in K, m \in M \rangle$ by $k \otimes m$. Next we consider the commutative monoid of finite bags of such pairs, with bag union as $+_{K \otimes M}$ and empty bag as $0_{K \otimes M}$. This forms a commutative monoid. Abusing notation, we will denote singleton bags by the unique element they contain. Then, every non-empty such bag can be written as $k_1 \otimes m_1 +_{K \otimes M} \cdots +_{K \otimes M} k_n \otimes m_n$ (with possible repetitions).

Let $\sim$ be the smallest congruence w.r.t. $+_{K \otimes M}$ that satisfies (for all k, k', m, m'):

$$0_K \otimes m \sim 0_{K \otimes M}$$
$$k \otimes (m +_M m') \sim k \otimes m +_{K \otimes M} k \otimes m'$$
$$k \otimes 0_M \sim 0_{K \otimes M}$$

We denote by $K \otimes M$ the set of *tensors*, i.e., equivalence classes of bags of $k \otimes m$ elements modulo $\sim$. Note that the axiom of associativity w.r.t. K from [2] is omitted here.

5.2 Annotated Relations

The notion of semiring-annotated relations was introduced in [20]. Given a commutative semiring K, a *K-relation* is essentially a relation whose tuples are associated with elements of K serving as their *annotations*. Formally, fix a countably infinite *domain* $\mathbb{D}$ of values (constants). For any finite set U of attributes, a tuple is a function $t : U \to \mathbb{D}$ and we denote the set of all such possible tuples by $\mathbb{D}^U$. A *K-relation* (with schema U) is then a function $R : \mathbb{D}^U \to K$ whose *support* $\mathrm{supp}(R) = \{t \mid R(t) \neq 0_K\}$ is *finite*, so that the *annotation* of tuple t in R is simply $R(t)$. Given a relational schema, K-databases are defined from K-relations just as relational databases are defined from usual relations. Note that $\mathbb{B}$-relations are just the standard (set) relations. Also, a semiring homomorphism $h : K \mapsto K'$ can be used to "switch" from a K-relation R to a K'-relation R', by replacing every annotation k with $h(k)$. Abusing notation we will use $h(R)$ to denote this transformation.

As shown in [20] there is a close correspondence between the semiring operations and the transformation of data. The $+$ operation on annotations corresponds to *alternative use* of data, the $\cdot$ operation to *joint use* of data, 1 annotates data that is always and unrestrictedly available, and 0 annotates absent data. This intuition is formalized in [20], where the operations of the positive relational algebra were extended to work on K-relations (as input and output). For example, *joining* two K-relations R, S results in a K-relation where each "joint" tuple obtained as a combination of a tuple t in R and t' in S is annotated with the annotation of t in R multiplied by the annotation of t' in S, namely $R(t) \cdot S(t')$, intuitively indicating that t and t' were used jointly to derive the new tuple. Similarly, projection leads to *summation* since it involves *alternative ways* of generating a tuple.

More generally, a *semantics* for an algebra on K-relations associates to each query in the algebra a mapping from K-relations to K-relations. For a query Q and an K-relation R we denote as $Q(R)$ the result of evaluating Q on R following the semantics. The semantics proposed in [20] satisfies the following fundamental properties:

Set-compatibility. When $K = \mathbb{B}$, the semantics coincides with the set-semantics for the algebra.[4]

Commutation with homomorphisms. For every two semirings K, K', query Q in the algebra, K-relation R, and homomorphism $h : K \mapsto K'$, we have $h(Q(R)) = Q(h(R))$.

[4]We discuss *bag*-compatibility at the end of Section 5.5

Poly-size overhead. For every query Q in the algebra and for every K-relation R, the size of $Q(R)$ (including annotations) is polynomial in the size of R.

Intuitively, the first two properties justify the correctness of the construction, in the sense that (i) it is a sound generalization of the semantics without annotations and (ii) it allows to "switch" between annotation domains, which is fundamental for provenance applications such as hypothetical reasoning [13]. The third property requires that the overhead in tracking provenance is not too large.

We next pursue the definition of a semantics for OWALG on annotated relations, using these properties as yardsticks.

5.3 A New Provenance Structure

A provenance construction for OWALG must address two novel challenges. The first is *nondeterminism of the query*, which, to our knowledge, has never been studied in this context. A second challenge is deciding which values should be used in the **Ord** attribute, and their interaction with nondeterminism and tuple annotations.

We will propose a novel structure for provenance for OWALG. Before presenting it, we motivate it by observing that even without non-determinism, K-relations are insufficient. The deterministic fragment of OWALG contains only queries for which there is exactly one possible world (when evaluated with respect to a "regular", non-annotated database).

PROPOSITION 5.1. *There is no semantics on K-relations for the deterministic fragment of* OWALG *that satisfies set-compatibility and commutation with homomorphisms.*

The proof is by a simple adaptation of a similar result in [2]. One may then ask whether the construction of [2] could be used instead. As we shall see, it will indeed be useful, but insufficient by itself.

We therefore construct a provenance structure as follows. Let K be a commutative semiring and let X be a set of "fresh" indeterminates (tokens), different from the tokens of K. Intuitively, we will use tokens from K for annotations and tokens from X to express conditions and deferred nondeterministic choices. Some care is then needed in the construction, to guarantee their correct interplay. We will assume that the set X is infinite so that we can always generate a new token in the construction. This is novel in the context of provenance tracking based on algebraic expressions. To capture conditions, we define $\mathrm{comp}[X]$ as the set of all *comparison expressions* involving polynomials over X, i.e., $\mathrm{comp}[X] = \{p \,\mathrm{op}\, p' \mid p, p' \in \mathbb{N}[X]\}$, where $\mathrm{op} \in \{=, \neq, <, \leqslant\}$. Note that $\mathbb{N}[X]$ includes in particular the natural numbers, so equations may involve e.g. elements of X on one side and natural numbers on the other side. Intuitively, these equations will be used as "abstract conditions", which may be "solved" under a *valuation* for X (defined below).

We then recall that $\mathbb{N}[T]$ is the semiring of polynomials with natural numbers as coefficients over a set T of indeterminates, and consider the semiring $\mathbb{N}[K \cup \mathrm{comp}[X]]$. that combines a given commutative semiring K and conditions over a given set X (i.e., symbolic indeterminates).

EXAMPLE 5.2. *If $x, y, z \in X$ then $[x + y \leqslant 7]$, and $[x^2 + 2xy = zx]$ are in* $\mathrm{comp}[X]$.

For $K = \mathbb{B}$, $\bot + \top \cdot [x + y \leqslant 7]$ is an element of $\mathbb{N}[K \cup \mathrm{comp}[X]]$. Similarly if $K = \mathbb{N}[\mathrm{Ann}]$ for some set $\mathrm{Ann} = \{a_1, a_2, \ldots\}$ of (annotation) indeterminates then $a_1 + a_2 \cdot [x^2 + 2xy = zx]$ is in $\mathrm{comp}[X]$.

While it may seem unnatural to thus combine elements of K and elements of $\mathrm{comp}[X]$, the intuition is that once elements of X are mapped to values, elements of $\mathrm{comp}[X]$ will be mapped to 0_K or 1_K.

We finally define a semiring $K(X)$ by taking the quotient on $\mathbb{N}[K \cup \mathrm{comp}[X]]$ by the smallest congruence satisfying the following axioms:

1. $0_{K(X)} \equiv 0_K$ and $1_{K(X)} \equiv 1_K$
2. For each $k_1, k_2 \in K$ it holds that $k_1 \cdot_{K(X)} k_2 \equiv_{K(X)} k_1 \cdot_K k_2$, and $k_1 +_{K(X)} k_2 \equiv_{K(X)} k_1 +_K k_2$
3. For each $k \in K, n \in \mathbb{N}$ it holds that $n \cdot k \equiv_{K(X)} k + \cdots + k$ (n times), and $k^n \equiv_{K(X)} k \cdots k$ (n times).

Valuations and Homomorphisms. The idea of "evaluating" the tokens is formalized by *valuations* for X. A *valuation*[5] for a set of tokens X is a function $V : X \to \mathbb{N}$. Note that such a function lifts naturally to a homomorphism $h_V : \mathbb{N}[X] \to \mathbb{N}$, which essentially replaces every occurrence of $x \in X$ in the polynomial by $V(x)$, and then simplifies with arithmetics on natural numbers. Finally we can further lift a valuation to a *homomorphism $\hat{h}_V$ from $K(X)$ to K* as follows. We define the result of applying $\hat{h}_V$ to equation elements $[p_1 \,\mathrm{op}\, p_2]$ as $\hat{h}_V([p_1 \,\mathrm{op}\, p_2]) = 1_{K(X)}$ if $h_V(p_1) \,\mathrm{op}\, h_V(p_2)$ holds (which only involves comparing natural numbers), and $\hat{h}_V([p_1 \,\mathrm{op}\, p_2]) = 0_{K(X)}$ otherwise (this is well-defined, even under the congruences). For $k \in K$, we define $\hat{h}_V(k) = k$ and then extend $\hat{h}_V$ to a homomorphism on $K(X)$ as $\hat{h}_V(exp_1 \cdot exp_2) = \hat{h}_V(exp_1) \cdot \hat{h}_V(exp_2)$ and $\hat{h}_V(exp_1 + exp_2) = \hat{h}_V(exp_1) + \hat{h}_V(exp_2)$.

Note that applying $\hat{h}_V$ has the effect of evaluating all $\mathrm{comp}[X]$ subexpressions to elements of K, so the result is in $\mathbb{N}[K]$, quotiented by the axioms above, and this structure is isomorphic to K (intuitively, exponents and multiplications by coefficients can be computed as elements of K by axiom 3, and then terms may be combined by axioms 1-2).

In addition, the standard way to "switch" between semirings is through a semiring homomorphism, and we can easily lift such homomorphisms $h : K \mapsto K'$ to $\hat{h} : K(X) \mapsto K'(X)$ which simply maps every occurrence of $k \in K$ to $h(k)$.

EXAMPLE 5.3. *Consider $a_1 \cdot [x + y \leqslant 3] + a_2 \cdot [x + y \leqslant 7]$, which is in $(\mathbb{N}[Ann])(X)$. Given a valuation such as $V(x) = 3$, $V(y) = 2$ we have $h_V(x + y) = 5$, $\hat{h}_V([x + y \leqslant 3]) = 0_K$, and $\hat{h}_V([x + y \leqslant 7]) = 1_K$. We then have $\hat{h}_V(a_1 \cdot [x + y \leqslant 3] + a_2 \cdot [x + y \leqslant 7]) = a_1 \cdot 0_K + a_2 \cdot 1_K = a_2$ where the last equality is by applying the congruences.*

Tensors. $K(X)$ provides a way to combine annotations and conditions over non-deterministic choices. The last "ingredient" required is the ability to pair such combinations with

[5] We restrict valuations to be only to natural numbers, due to the particular way we use elements of X.

natural numbers, to allow representing order in a provenance-aware fashion. This is done via the *tensor product* with the monoid of natural numbers $\mathbb{N}$ with standard $+_{\mathbb{N}}$ and $0_{\mathbb{N}}$.

Annotated Relations. In our provenance construction, we will use $K(X)\otimes\mathbb{N}$ as *domain* of the **Ord** attribute (and only **Ord**). By contrast, the annotations of tuples will be elements of some $K(X)$. We call such relations (resp. databases) $(K(X)\otimes\mathbb{N},K(X))$-*relations (resp. databases)*. We continue to use K-relations to denote relations with tuples are annotated by elements of K, and with no annotations in values. We then define the effect of homomorphisms and valuations on such relations in the natural way: given a $(K(X)\otimes\mathbb{N},K(X))$ database D and a valuation $V : X \mapsto \mathbb{N}$ we use $V(D)$ to denote the $(K\otimes\mathbb{N},K)$-database obtained by replacing every element *exp* of $K(X)$ occurring in values or annotations of D, by $V(exp)$. Similarly, applying a semiring homomorphism $h : K \mapsto K'$ to a $(K(X)\otimes\mathbb{N},K(X))$-database (resp. $(K\otimes\mathbb{N},K)$-database) yields a $(K'(X)\otimes\mathbb{N},K'(X))$-database, (resp. a $(K'\otimes\mathbb{N},K')$-database).

We need one final construct that will allow us to "read" elements in $K\otimes\mathbb{N}$ as natural numbers. For that, when $K=\mathbb{B}$ (the boolean semiring) we define the embedding function on $\mathbb{B}\otimes\mathbb{N}$ by $\iota(\top\otimes n)=n$ and $\iota(\bot\otimes n)=0$, and again extend it to a homomorphism and to relations with the semantics that it is applied on any element of the form $b\otimes n$ in the relation.

Now that we have defined the proper algebraic structure, we are finally ready to define provenance for OWALG. This is done in two steps. We first define the construction for each order-aware operator (Rank, Concat, Shuffle, Sync) individually. We assume that the operator is the last to be applied (following any combination of SPJ operators). Except for Rank, we also assume the operator is applied on already ordered relations. After studying the properties of the construction in Section 5.5, we will extend it in Section 5.6 to account for any query in OWALG, in particular allowing the composition of order-aware operators. We will assume, throughout the construction, that X is a fixed infinite set of variables.

5.4 Provenance for Order-Aware Operators

We define the semantics of each operator as a mapping from K-relations to $(K(X)\otimes\mathbb{N},K(X))$-relations.

Rank. Recall that the Rank operator generates an ordered relation out of an unordered one, by introducing a new **Ord** column, whose (numerical) values reflect the order of tuples. For annotated relations, we need to reflect, as part of the order of a tuple, the annotations of other tuples. The idea is that we generate an expression combining value and annotations, that "abstractly counts" the number of tuples with smaller values in the column, which is made possible by the tensor product. This is in fact equivalent to the result of a corresponding count query according to the semantics of [2].

DEFINITION 5.4. *Let K be a commutative semiring and let R be a K-relation, $A\subseteq\text{attr}(R)$. We define* $\text{Rank}_{A,<}(R)$ *as a relation S, with:*

- $\text{attr}(S)=\text{attr}(R)\cup\{\textbf{Ord}\}$.
- $\Pi_{-\textbf{Ord}}(S)=\Pi_{-\textbf{Ord}}(R)$.
- *For each $s\in S$, $S(s)=R(\Pi_{\text{attr}(R)}(s))$.*
- *For each $\in S$, $s.\textbf{Ord}=\sum_{\{t'\in\text{supp}(R)|t'.A<t.A\}}R(t')\otimes 1_{\mathbb{N}}$ where we sum in $K(X)\otimes\mathbb{N}$, and $1_{\mathbb{N}}$ is the integer 1.*

Note that Rank does not yet use the set X; it will only be needed for the nondeterministic operations.

EXAMPLE 5.5. *Reconsider relation R_1 in Table 1.a (without the* **Ord** *attribute). further consider an $\mathbb{N}[Ann]$-relation R_{ann} with same tuples as R_1 and annotated with $Ann=\{r_0,r_1,...\}$ s.t. the annotation of tuple with id Ri is r_i.*

Also reconsider the query $\text{Rank}_{Time,<}(R)$*. Its (provenance-aware) result is (the* **Ann** *column indicates the annotations of tuples and is not part of the relation):*

ID	Read	Loc	Time	**Ord**	**Ann**
R0	Light	1	3:30:02	0	r_0
R1	Door	1	3:30:04	$r_0\otimes 1+r_2\otimes 1$	r_1
R2	Light	3	3:30:03	$r_0\otimes 1$	r_2

Intuitively, the **Ord** *value of e.g. the* R2 *tuple adds a count of 1 "paired" with r_0 (intuitively due to the r_0-annotated tuple) and another paired with r_2. Intuitively*[6]*, we may then consider hypothetical scenarios for the presence or absence of each tuple (a tuple may e.g. be considered absent if it is untrusted), express them as a homomorphism h from $\mathbb{N}[Ann]$ to $\mathbb{B}$, and lift it to both annotations and* **Ord** *values. E.g. if $h(r_0)=0$, $h(r_1)=h(r_2)=1$, the annotation of tuple R0 is* 0*, i.e. it is discarded. For R1 in contrast we get* 1 *(i.e. it is present) and its* **Ord** *is $0\otimes 1+1\otimes 1$, which via ι is interpreted as* 1 *as, in this scenario, there is just one tuple (R2) that precedes it.*

Concat. The next operator to consider is Concat, with a similar idea for provenance propagation: the rank of an item is "parameterized" by provenance tokens indicating the existence of other tuples that may precede it.

DEFINITION 5.6. *Let R,R' be two K-relations. We define* $\text{Concat}(R,R')=S$ *such that* $\text{attr}(S)=\text{attr}(R)$*, and there is an isomorphism h between $R\cup R'$ (bag union) and S, satisfying:*

- *For all $s\in S$, $\Pi_{-\textbf{Ord}}(s)=\Pi_{-\textbf{Ord}}(h^{-1}(s))$.*
- *For all $s\in S$, $S(s)=R(r)$ or $R'(r')$ depending on whether $h^{-1}(s)$ is in R or R'.*
- *If $s=h(r)$ s.t. $r\in R$, then we can define $s.\textbf{Ord}=\sum_{\{r_1\in R|r_1.\textbf{Ord}<r.\textbf{Ord}\}}(R(r_1)\otimes 1_{\mathbb{N}})$*
- *If $s=h(r')$ s.t. $r'\in R'$, then $s.\textbf{Ord}=\sum_{\{r_1\in R\}}(R(r_1)\otimes 1_{\mathbb{N}})+\sum_{\{r'_1\in R'|r_1.\textbf{Ord}<r.\textbf{Ord}\}}(R'(r'_1)\otimes 1_{\mathbb{N}})$*

Shuffle. We next consider Shuffle. Its nondeterminism poses further challenges in the construction. Let R and R' be two ordered K-relations over the same schema and assume w.l.o.g.[7] that the **Ord** attribute of R (resp. R') contains contiguous values $0,...,n$ (resp. $0,...,m$). Let r_i (r'_i) be the tuple in R (resp. R') with **Ord** value i.

Further let $Y=\{Y_0,Y_1,Y_2,...,Y_n\}$ be a set of fresh variable names from X. Intuitively, $\sum_{j\leqslant i}Y_j$ indicates how many tuples of R' appear before r_i.

[6]Using commutation with homomorphisms, shown in Section 5.5.

[7]See discussion in Section 2, and the construction for nested operations in Section 5.6

We can now define the semantics of Shuffle for annotated relations. Let S be the output of Shuffle(R,R'). For each $r_i \in R$ there is a tuple s_i in S. The attribute **Ord** is defined as:

$$s_i.\mathbf{Ord} = \sum_{j<i}(R(r_j)\otimes 1_{\mathbb{N}}) + \sum_{j\leqslant m}\left(\left(R'(r_j)\cdot\left[j<\sum_{k\leqslant i}Y_k\right]\right)\otimes 1_{\mathbb{N}}\right)$$

Intuitively, the first term counts the number of tuples of R preceding r_i, and the second one counts the number of tuples of R' preceding r_i according to the chosen valuation of X. Note the subtlety in using the annotation of tuples as part of the expression: this is to account for their possible mapping to 0 (see discussion on set compatibility below). The other attributes of s_i have the same values as those of r_i, and likewise $S(s_i) = R(r_i)$.

Similarly, for each $r'_i \in R'$, we have a tuple s'_i in S s.t.:

$$s'_i.\mathbf{Ord} = \sum_{j<i}(R'(r'_j)\otimes 1) + \sum_{j\leqslant n}\left(\left(\left[\sum_{m\leqslant j}Y_m \leqslant i\right]\cdot R(r_j)\right)\otimes 1\right)$$

The other attributes of s'_i have the same values as those of r'_i. The annotation associated with the tuple s_i (s'_i) is the same as the annotation of r_i in R (respectively r'_i in R').

EXAMPLE 5.7. *Reconsider Example 2.7, assuming now that tuples R0, R1, R2, T0, T1 are annotated with r_0, r_1, r_2, t_0, t_1 respectively. The annotated* Shuffle *result is then:*

Read	**Ord**	**Ann**
Light	$(t_0\cdot[0<Y_0])\otimes 1+(t_1\cdot[1<Y_0])\otimes 1$	r_0
Door	$(t_0\cdot[0<Y_0+Y_1])\otimes 1+(t_1\cdot[1<Y_0+Y_1])\otimes 1$	r_1
Light	$(t_0\cdot[0<Y_0+Y_1])\otimes 1+(t_1\cdot[1<Y_0+Y_1])\otimes 1+(t_2\cdot[1<Y_0+Y_1+Y_2])\otimes 1$	r_2
Beep	$(r_0\cdot[Y_0\leqslant 0])\otimes 1+(r_1\cdot[Y_0+Y_1\leqslant 0])\otimes 1+(r_2\cdot[Y_0+Y_1+Y_2\leqslant 0])\otimes 1$	t_0
Door	$t_0\otimes 1+(r_0\cdot[Y_0\leqslant 1])\otimes 1+(r_1\cdot[Y_0+Y_1\leqslant 1])\otimes 1+(r_2\cdot[Y_0+Y_1+Y_2\leqslant 1])\otimes 1$	t_1

Now, one may use a valuation of the Y variables to choose a possible world for the Shuffle*: each Y_i is used to decide how many elements of S are to be placed between R_{i-1} and R_i (Y_0 is used to choose how many will precede R_0). For instance by choosing $Y_0 = 1, Y_1 = Y_2 = 0$ we obtain the possible world in Table 1.d. Furthermore, a scenario where some of the tuples are omitted (e.g. as they are untrusted) may be expressed as a homomorphism mapping r_0, r_1, r_2, t_0, t_1 to truth values; lifting this homomorphism to the annotated result allows exactly all valid orderings of the tuples that remain.*

Sync. Last, we define provenance for the Sync operation. We introduce one new variable $Y_t \in X$ for every tuple t in the *set* union of $\Pi_{-\mathbf{Ord}}(R\cup R')$. Then, given two ordered relations R and R' with same schema we define Sync(R,R') as a relation S, as follows. Projected on attr$(R)-\{Ord\}$, the tuples of S are exactly the tuples of $\Pi_{-\mathbf{Ord}}(R\bigcup R')$, and the **Ord** attribute of a tuple t is defined as:

$$t.\mathbf{Ord} = \sum_{t'\in R\cup R'}([Y_{t'} < Y_t]\otimes 1_{\mathbb{N}})$$

Note that so far the construction ignores the original orders in R and S. These are accounted for by the expression:

$$G = \Pi_{\{t,t'\in R\cup R'\mid \text{precedes}(t,t')\}}[Y_t < Y_t]\cdot\Pi_{\{t\neq t'\in R\cup R'\}}[Y_t \neq Y_{t'}]$$

where precedes(t,t') holds if and only if t must precede t' by the definition of the (un-annotated) Sync operator.

We then set the *provenance* of every tuple t to be $(R(t)+R'(t))\cdot G$. Thus, a valuation to the Y_t's must respect the constraints imposed by G (agree with the original orders and not assign the same location to two different tuples), or it will lead to the empty relation (which is a possible world).

5.5 Properties of the Construction

Putting together the above definitions, with the semantics for SPJ queries on K-relations, we obtain a semantics for a fragment of OWALG (denoted in the sequel OWALG$'$), where we apply only one order-aware operator, and do so as the last operation (if Concat, Shuffle, Sync is used then the input relation should already be ordered, that is, contain an **Ord** attribute). This assumption is relaxed in Section 5.6, where we consider the full language, but we can already examine the usefulness of this simplified construction.

We therefore consider the desiderata for provenance constructions, and check that our construction satisfies them.

Set-compatibility. Recall that in our construction (unlike previous provenance constructions for deterministic query languages), there are two domains for which mappings may be defined: the semiring K and the set of indeterminates X. This affects the formulation of set-compatibility. Intuitively, we claim that, for the Boolean semiring $K = \mathbb{B}$. the valuations for X allow to choose exactly from the possible worlds of the nondeterministic query result.

To formalize the claim, recall that for a $\mathbb{B}$-database D, supp(D) is the (regular) database consisting of only the tuples of D annotated with non-zero (in this case, $\top$) annotations. Further recall the definition of valuation V to variables and the embedding ι of tensors into natural numbers, as well as their extensions to application on databases. We have:

PROPOSITION 5.8. *Let $Q \in$ OWALG$'$ and let D be a $\mathbb{B}$-database. The following holds:*

$$Q(\text{supp}(D)) = \{\text{supp}(\iota(V(Q(D)))) \mid V : X \mapsto \mathbb{N}\}$$

Intuitively, $Q(\text{supp}(D))$ is the query result according to the semantics in Section 2, on a non-annotated database obtained from D by keeping only the tuples in its support. The proposition means that this result (which is a set of possible worlds, since Q is nondeterministic) is exactly the set of possible worlds one may obtain by (1) computing the annotated output $Q(D)$, (2) giving a valuation to the X variables (and solving equation elements) and mapping tensors to natural numbers to obtain an $\mathbb{B}$-database D', and (3) generating a non-annotated database by keeping only the tuples of D' that are not annotated with 0.

Note that this correctly generalizes set-compatibility for deterministic queries: if X is empty there is exactly one world, and we get $Q(\text{supp}(D)) = \text{supp}(\iota(Q(D)))$.

Commutation With Homomorphisms. The next property is commutation with semiring homomorphisms, which holds for our semantics:

PROPOSITION 5.9. *Let K, K' be commutative semirings, let $h : K \mapsto K'$ be a semiring homomorphism, let D be a K-database and let $Q \in$ OWALG$'$. We have $h(Q(D)) = Q(h(D))$.*

We have already exemplified above the effect of applying homomorphisms (using this property implicitly). We next show another example, in the context of *access control*.

EXAMPLE 5.10. *Consider the $\mathbb{S}$ semiring [20] and relation R of Example 5.5, but now replace r_0, r_1, r_2 by $1_\mathbb{S}, \mathsf{C}, \mathsf{S}$ (public, confidential, secret). This means that some observations are confidential and may only be revealed to users with sufficient clearance. Now reconsider the query* $\mathrm{Rank}_{Time,<}(R)$. *Its (provenance-aware) result is:*

ID	Read	Loc	Time	**Ord**	**Ann**
R0	Light	1	3:30:02	0	$1_\mathbb{S}$
R1	Door	1	3:30:04	$1_\mathbb{S} \otimes 1 + \mathsf{S} \otimes 1$	C
R2	Light	3	3:30:03	$1_\mathbb{S} \otimes 1$	S

Now every given clearance CL defines a homomorphism from the access control semiring to $\mathbb{N}$, in the following simple way: if $k >$ CL then $h(k) = 0$, and otherwise $h(k) = 1$. So, for instance, for a user with "confidential" clearance, we would map both P and C to 1 and S to 0. Commutation with homomorphisms tells us we can apply this directly to the annotated query result, to obtain order 0 and 1 for thhe R0 and R1 tuples, and to omit R2 since its annotation is 0.

Poly-size Overhead. We finally show (proof in the appendix) that poly-size overhead holds:

PROPOSITION 5.11. *Let D be a K-relation and let $Q \in$ OWALG$'$. The size of $Q(D)$ is polynomial in that of D.*

Considering Bag-compatibility. Having focused on set-compatibility, we conclude the discussion by pointing out the fundamental difficulty in achieving a semantics that commutes with homomorphisms and is bag-compatible (i.e. coincides with the bag-semantics when applied on $\mathbb{N}$-relations).

EXAMPLE 5.12. *Reconsider Example 2.15, but now assume that instead of multiplicities, we use abstract annotations x and y for the multiplicities of a in R and b in R' respectively (so they are now $\mathbb{N}$-relations). Now for every n, let h_n be the valuation mapping x and y to n. For the result of Q on R, R' to commute with homomorphisms, it must have $2n$ tuples (to account for the possible world of alternating a's and b's). However, the size of the provenance representation should not depend on the (yet unknown) n.*

This indicates that a bag-compatible provenance tracking would require a radically different approach. This is an intriguing task for future work.

5.6 Compositions of Operators

We have defined provenance-aware semantics for each of the order-aware operators when applied to a K-relation. However nested application of order-aware operators is not well-defined yet, as the output of, say, Shuffle(R), is not a K-relation but a $(K(X) \otimes \mathbb{N}, K(X))$-relation so it does not match the expected input type of the operators. We can handle the full language with the following modifications:

Provenance Structure. We refine the construction of the structure as follows. Essentially, we need to be able to *nest* comparison expressions and tensors at any depth. This means that tensors as well as comparison expressions may be used inside comparison expressions. We thus refine the definition of comp$[\cdot]$ so that it may be applied to the elements of arbitrary tensor products $K \otimes M$ with K a commutative semiring and M a commutative monoid. In particular, this allows the comparisons over tensors required for nested queries. For $K = \mathbb{B}$, we impose the additional congruences $[k_1 \otimes m_1 = k_2 \otimes m_2] \equiv 1$ when $\iota(k_1 \otimes m_1) = \iota(k_2 \otimes m_2)$, and equivalent to 0 otherwise.

We then let $K(X)_{nest}$ be the least solution (well-defined as the right hand side is monotone) of the domain equation:

$$K(X)_{nest} = K(X) \cup \mathrm{comp}[K(X)_{nest} \otimes \mathbb{N}]$$

Semiring homomorphisms and indeterminates valuations may be applied as usual, replacing every $k \in K$ with $h(k) \in K'$ and $x \in X$ with $v(x)$. To "solve" these equation elements (i.e. map them to 1_K or 0_K) we need to be able to compare tensors (and tensors with values). For that we use ι.

More concretely, let $K = \mathbb{B}$ and let $V : X \mapsto \mathbb{N}$ be a valuation. Further let $exp \in K(X)_{nest}$ be an expression of nesting depth i (i.e. it is expressible in the domain obtained by i iterations of the fixpoint equation). We define $v(exp)$ via recursion: If $i = 0$ then $exp \in K(X)$ and $V(exp) \in K$ is defined as above. Otherwise by the recursive construction, V may be applied to every exp' such that $exp' \otimes n$ occurs in an equation in exp. By lifting it we have $V(exp') \otimes n \in K \otimes \mathbb{N}$, and by noting that $K = \mathbb{B}$ and using the congruences above, we obtain an expression in K.

With this definition, we can apply valuations and homomorphisms to $(K(X)_{nest} \otimes \mathbb{N}, K(X)_{nest})$-relations as before.

Semantics of Operators. We can now build, using $K(X)_{nest}$, a semantics satisfying the three desiderata of set-compatibility, commutation with homomorphisms and poly-size overhead, defined as in Propositions 5.8, 5.9, and 5.11.

THEOREM 5.13. *There exists a semantics for* OWALG *on $(K(X)_{nest} \otimes \mathbb{N}, K(X)_{nest})$-relations which satisfies set-compatibility, poly-size overhead, and commutation with homomorphisms.*

6. RELATED WORK

We complete the discussion of related work, in addition to the discussion in the Introduction.

Queries on Ordered domains. In the context of the expressive power of queries, the effect of having total ordering on the universe has been studied extensively [22, 32]. In applications such as those we have considered, one may not be able to assume that a total order is available. Data transformation in presence of partial order was also studied in several contexts [27, 9, 4, 25, 21]. As mentioned in the Introduction, approaches differ in the way they address the consolidation of data; but we are not aware of a query language that includes explicit *non-deterministic operators* for consolidating the data, in conjunction with deterministic operators such as SPJ. Instead, some works choose one possible solution or forbid conflicts, etc. Others, such as e.g. [21] focus on manipulating *ordered types* (in this case partially

ordered multisets). As indicated by our expressiveness results there are some connections between the approaches, and e.g. our Concat and Shuffle operations for relations have counterparts as operations on partial orders. Two major conceptual differences between the approaches are (1) our operational approach leading to the non-deterministic semantics and (2) our interest in capturing the order at the level of *particular tuples*, rather then capturing an abstract representation of the order as a whole. These two choices have the benefit of allowing order-aware transformations to be easily expressed alongside with "traditional" querying. Even more importantly, our approach lends itself much more naturally to provenance tracking (which was not considered in any of these previous works). In particular, it allows our provenance construction to build upon, and extends known principles of provenance tracking for data transformations.

On the other hand, our approach leads to some limitations in expressiveness w.r.t. the manipulation of abstract ordered data types. A particular example is in the handling of multiplicities: in [21], the result of "shuffling" n copies of a elements and n copies of b elements can be kept as an abstract representation (the disjoint union of total orders on the a's and b's). In contrast, as we are interested in the location of each particular such a, we need to maintain its order individually, leading to the blowup observed in Example 2.15. Since our language is still quite expressive – we are able to capture all posets (rather than all partially ordered *multisets* in [21]) – we believe that it constitutes a reasonable tradeoff between expressiveness on the one hand and usability and provenance tracking on the other hand.

Provenance. Recording provenance information for query results, to explain the computational process leading to their generation, is now a common technique (see also e.g. [18, 14, 8, 12, 10, 6]) with applications such as view maintenance, trust assessment, or hypothetical reasoning. Some techniques employed here for provenance construction, such as the use of tensors for counting and ideas behind the nested construction are based on those employed in [2] for aggregate queries. The treatment of *non-determinism*, as well as its *interplay with annotations and values* are the main novel challenges that were addressed here. To our knowledge, provenance management of the flavor studied here was neither previously studied for order-aware transformation nor in the context of any non-deterministic query language. Our approach may also serve a first step towards the development of provenance frameworks for other non-deterministic query languages, such as languages with a repair operator [19].

7. CONCLUSION

We have proposed in this paper a non-deterministic language for querying relational databases in presence of order. We have exemplified the usefulness of the language, studied both its expressiveness and its complexity in terms of deciding whether a given instance is a possible world. We have further introduced a semiring-based provenance framework for the language. There are many intriguing directions that we intend to pursue as future work, such as: extending the approach to nested relations and XML; further refining the tractability bounds for complexity; developing of optimization techniques to further reduce provenance size in practice; and implementing software prototypes using the framework.

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

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, 2011.
[3] D. Bechet, P. d. Groote, and C. Retoré. A complete axiomatisation for the inclusion of series-parallel partial orders. In *RTA*, 1997.
[4] M. Benedikt and C. Koch. XPath leashed. *ACM Comput. Surv.*, 41(1), 2008.
[5] O. Benjelloun, A. Sarma, A. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB J.*, 17, 2008.
[6] S. C. Boulakia and W. C. Tan. Provenance in scientific databases. In *Encyclopedia of Database Systems*, pages 2202–2207. 2009.
[7] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Posets*, chapter 6. SIAM, 1987.
[8] P. Buneman, J. Cheney, and S. Vansummeren. On the expressiveness of implicit provenance in query and update languages. *ACM Trans. Database Syst.*, 33(4), 2008.
[9] P. Buneman, A. Jung, and A. Ohori. Using powerdomains to generalize relational databases. *Theor. Comput. Sci.*, 91(1), 1991.
[10] P. Buneman, S. Khanna, and W. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.
[11] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.
[12] Y. Cui, J. Widom, and J. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2), 2000.
[13] D. Deutch, Z. Ives, T. Milo, and V. Tannen. Caravan: Provisioning for what-if analysis. In *CIDR*, 2013.
[14] R. Fink, L. Han, and D. Olteanu. Aggregation in probabilistic databases via knowledge compilation. *PVLDB*, 5(5), 2012.
[15] J. N. Foster, T. J. Green, and V. Tannen. Annotated XML: Queries and Provenance. In *PODS*, 2008.
[16] M. R. Garey and D. S. Johnson. *Computers And Intractability. A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.
[17] F. Geerts, G. Karvounarakis, V. Christophides, and I. Fundulaki. Algebraic structures for capturing the provenance of SPARQL queries. In *ICDT*, 2013.
[18] F. Geerts and A. Poggi. On database query languages for k-relations. *J. Applied Logic*, 8(2), 2010.
[19] M. Götz and C. Koch. A compositional framework for complex queries over uncertain data. In *ICDT*, 2009.
[20] T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.
[21] S. Grumbach and T. Milo. An algebra for pomsets. In *ICDT*. 1995.
[22] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1-3), 1986.
[23] *ISO 9075:2008: SQL*. International Standards Organization, 2008.
[24] R. Kohli, R. Krishnamurti, and P. Mirchandani. The minimum satisfiability problem. *SIAM J. Discret. Math.*, 7(2), 1994.
[25] T. Lindholm. A three-way merge for XML documents. In *DocEng*, 2004.
[26] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1), 2010.
[27] W. Ng. An extension of the relational data model to incorporate ordered domains. *ACM TODS*, 26(3), 2001.
[28] B. Schröder. *Ordered Sets: An Introduction*. Birkhäuser, 2003.
[29] M. A. Soliman, I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi. Ranking with uncertain scoring functions: semantics and sensitivity measures. In *SIGMOD*, 2011.
[30] B. Sundararaman, U. Buy, and A. D. Kshemkalyani. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 3(3), 2005.
[31] S. Vansummeren and J. Cheney. Recording provenance for SQL queries and updates. *IEEE Data Eng. Bull.*, 30(4), 2007.
[32] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, 1982.

APPENDIX

A. PROOFS FOR SECTION 2

A.1 Proof of Proposition 2.12

We use $\simeq$ as shorthand like in Definition 2.9.

It suffices to show that $\text{Sync}(R,R')$ has a possible world. Let $t_1,\ldots,t_n$ be a sequence enumerating a subset of the tuples of R (in their order in R) satisfying that for each t_i, there is t'_i in R' such that $t_i \simeq t'_i$. (Possibly $n=0$ and we have no common elements). In this case, by the assumption that arguments of Sync have no duplicates-up-to-Ord , t'_i is unique, and there is no $i \neq j$ such that $t_i \simeq t_j$.

Because we assumed that we have no conflicts between R and R', then, whenever $i<j$ and thus t_i precedes t_j in R, we know that t'_i precedes t'_j in R'. So, in fact, we have $t_i \simeq t'_j$ if and only if $i=j$.

For $0 \leqslant i \leqslant n$, we write R_i the sequence of the tuples of R which follow t_i and precede t_{i+1} (ignoring the conditions of following the non-existing t_0 or preceding the non-existing t_{n+1}), in the order in which they appear in R. Define likewise R'_i for R'. By definition of the t_i, for all i,j, there is no $t \in R_i$ such that $t \simeq t'$ for some $t' \in R'_j$.

Define the table S by putting the tuples in the following order: $R_0,R'_0,t_1,R_1,R'_1,t_2,\ldots,t_{n-1},R_n,R'_n,t_n,R_{n+1},R'_{n+1}$. As R and R' are not empty, clearly S is not empty. We verify that S is a possible world of $\text{Shuffle}(R,R')$. The first condition is immediate. As for the second, it is easy to see that there are no two tuples $t_1,t_2 \in S$ such that t_1 precedes t_2 in S but t'_2 precedes t'_1 in one of R,R', $t'_1 \simeq t_1$, $t'_2 \simeq t_2$.

A.2 Proof of Proposition 2.13

We use $\simeq$ as shorthand like in Definition 2.9.

Consider $S=R$. We check that S is indeed a possible world of $\text{Sync}(R,R')$. The first condition is immediate. As for the second, notice that, if t_1 precedes t_2 in S and there are t'_1 and t'_2 in $T \in \{R,R'\}$ such that t'_2 precedes t'_1 and $t'_2 \simeq t_2$, $t'_1 \simeq t_1$, then necessarily (because there are no duplicates) $T=R'$; but then there are $t''_1=t_1$, $t''_2=t_2$ in $S=R$ such that t''_1 precedes t''_2 in R.

A.3 Proof of Proposition 2.14

See Definition 3.2 for definitions of the order-theoretic notions used here.

Consider $<'$ the partial order obtained by extending the total order $<$ on $\text{dom}(A)$ to $\text{dom}(R)$ (so that two tuples are comparable for $<'$ if and only if they have different values for attribute A).

Let $<_+$ be a linear extension of $<'$ on $\text{attr}(R)$ extending $<'$ so that, for any value of attribute A, $<_+$ is a total order on all possible tuples with this value in attribute A, and clearly those choices are independent (for every value of A, any total order could be chosen). Let $<_-$ be a linear extension of $<'$ on $\text{attr}(R)$ obtaining by choosing the *reverse* total order for each class. Hence, if t and t' have different values for attribute A, then either $t<t'$ and $t<_+t'$, $t<_-t'$ or $t'<t$ and $t'<_+t$, $t'<_-t$; if they do not, then t and t' are incomparable for $<'$ and either $t<_+t'$ and $t'<_-t$, or $t'<_+t$ and $t<_-t'$.

Let $R_+ = \text{Rank}_{\text{attr}(R),<_+}$ and $R_- = \text{Rank}_{\text{attr}(R),<_-}$; they are well-defined, because if $\pi_{\text{attr}(R)}(t) = \pi_{\text{attr}(R)}(t')$ then $t=t'$ because we are in the set semantics. Consider $R' = \text{Sync}(R_+,R_-)$. Because they have same domain, Sync succeeds according to Proposition 2.13.

Now, clearly $\pi_{\text{attr}(R)}(R') = R$ except for the added *Ord* attribute, and clearly if t_1 precedes t_2 in R then t_1 precedes t_2 in R' (because the corresponding t'_1 precedes the corresponding t'_2 in both R_+ and R_-), so every possible world of R' is a possible way of of breaking ties. Conversely, any way of breaking ties is a possible world of R', because, between two tuples t and t' tied in R because of an equal A-value, there is no order constraint in R' between the corresponding tuples, because R_+ and R_- must disagree on them. Hence, we have proved our result, up to the additional empty relation possible world introduced by the Sync.

B. PROOFS FOR SECTION 3

B.1 Proof of Theorem 3.6

First claim. We first show that, for any OWALG query Q without Sync and database D, if Q does not include a σ_{Ord} operator and $Q(D)$ is an ordered relation, then there is a series-parallel poset P of size polynomial in Q and D such that $Q(D)$ represents P.

Let us first show the following:

LEMMA B.1. *For any database D and* OWALG *query Q not using* Sync *or σ_{Ord}, (i) if $Q(D)$ is not ordered then it has exactly one possible world (ii) if $Q(D)$ is ordered then for any possible worlds W and W' of $Q(D)$ we have $\pi_{-\mathsf{Ord}}(W) = \pi_{-\mathsf{Ord}}(W')$.*

PROOF. We proceed by structural induction on Q depending on the last operator applied. The base case of raw relations is clear (they are unordered and have only one possible world):

Rank. By induction hypothesis, the unordered subexpression under consideration has only one possible world. No matter how we break ties, property (ii) holds.

Concat. By induction hypothesis, the subexpressions tables satisfy property (ii), so the Concat also satisfies it by definition.

Shuffle. By induction hypothesis, the two subexpressions satisfy property (ii), so the Shuffle also satisfies it by definition.

SPJ on unordered subexpression(s). By induction hypothesis, the subexpression(s) have only one possible world, and SPJ operators on unordered subexpression(s) are deterministic so the result is unordered and has exactly one possible world.

Projection keeping Ord. No tuples are eliminated when performing a projection on an ordered subexpressions which maintains the **Ord** column, because the **Ord** values ensure that no duplicates exist, so property (ii) is maintained.

Projection removing Ord. By property (ii), all the possible worlds of the ordered subexpression collapse to the same

unordered possible world, so the unordered result has exactly one possible world.

Selection on an ordered relation. As the selection is not on **Ord**, property (ii) is preserved because $\Pi_{-\mathbf{Ord}}(\sigma(A)) = \sigma(\Pi_{-\mathbf{Ord}}(A))$ if σ does not involve **Ord**.

Join on an ordered relation. As the join cannot involve **Ord** by definition of the join, property (ii) is preserved because $\Pi_{-\mathbf{Ord}}(A \bowtie B) = (\Pi_{-\mathbf{Ord}}(A)) \bowtie B$ (where A is the ordered and B the unordered argument to $\bowtie$) if $\bowtie$ does not involve **Ord**. Property (ii) is then preserved for all possible ways of breaking ties in **Ord** in the result.

□

We next define the notion of *restriction* of a (labeled) poset:

DEFINITION B.2. *A* restriction *of a partial order $P = (V, <)$ is a partial order $P' = (V', <')$ where $V' \subseteq V$ and $<'$ is the restriction of $<$ to $V' \times V'$. This definition extends to labeled posets.*

We show that the class of series-parallel posets is stable under restriction:

LEMMA B.3 ([3]). *Any non-empty restriction of a series-parallel (labeled) poset is still a series-parallel (labeled) poset.*

We now introduce another operation that we need to support:

LEMMA B.4. *Let $P = (V, <)$ be a series-parallel poset and consider $v \in V$. Let $X = \{x_1, \ldots, x_n\}$ be a set of fresh values, and define $P' = (V \cup X, <')$ as follows: for all $v' \in V$, $x \in X$, $v' <' x$ if and only if $v' < v$, and $x <' v'$ if and only if $v < v'$; for all $v', v'' \in V$, $v' <' v''$ if and only if $v' < v''$, and there is no $x, x' \in X$ such that $x <' x$ holds. Then P' is a series-parallel order.*

PROOF. Consider an expression of series and parallel compositions yielding P from singleton orders, and replace the singleton element $\{v\}$ by a parallel composition of singleton orders for $X \cup \{v\}$. It is clear that we obtain the prescribed order. □

We can now prove Theorem 3.6 by structural induction on Q. We do a case disjunction on the last operator applied.

Rank. If the last operator is Rank then by Lemma B.1 its unordered argument has exactly one possible world, so $Q(D)$ is a total order and it can be represented by a series-parallel labeled poset of size linear in Q and D.

Concat. If the last operator is Concat and both subexpressions represent labeled posets P_1 and P_2 respectively, the possible worlds of $Q(D)$ are the set of concatenations of linear extensions of P_1 and P_2, so that $Q(D)$ represents the series composition of P_1 and P_2. Likewise, if the last operator is Shuffle, $Q(D)$ represents the parallel composition of P_1 and P_2.

Projection. If the last operation is a projection, it cannot project out **Ord** as we assume the top-level expression is an ordered relation. So it projects out some other attribute, and the result is still representable by a labeled series-parallel poset (it amounts to relabeling the elements).

Selection. As selections cannot involve **Ord**, they involve some other attribute. Let P be the series-parallel labeled poset represented by the subexpression of the expression under consideration. By Lemma B.3, the restriction of P to remove the elements that are removed by the selection is either a series-parallel labeled poset or the empty poset.

Join. If the last operation is a join between an ordered relation and an unordered one, remark first by Lemma B.1 that the unordered relation has exactly one possible world. Let P be the poset represented by the ordered relation being joined. By Lemma B.3, the restriction of P to remove the elements for tuples that have been removed is still series-parallel (or is empty). Now, observe that the order between duplicate tuples in the result of the join is entirely unspecified, but their order relative to other tuples in the result is the same as that of the original tuple. We can therefore apply Lemma B.4 and conclude that the resulting order is still series-parallel. (The argument of the Lemma still applies to labeled posets.) As for the tuple values, this amounts to changing the label set and labeling function of the series-parallel poset.

Second claim. We now prove the second direction: for any product Σ of domains and series-parallel poset P on Σ, there exists an OWALG query Q without Sync and a database D of size polynomial in P such that $Q(D)$ represents P.

This claim is straightforward: consider the construction of P from single-element posets using the series and parallel composition, and, letting D be single-tuple relations matching the single-element posets, write Q to mimic the construction of P using Concat for series composition and Shuffle for parallel composition.

B.2 Proof of Theorem 3.9

First claim. We show that, for any OWALG query Q and database D, if $Q(D)$ is ordered and Q does not include a $\sigma_{\mathbf{Ord}}$ operator, then either $Q(D)$ has no possible world where no Sync operator fails or there is a poset P of size polynomial in Q and D such that $Q(D)$ represents P.

We first observe that the following variant of Lemma B.1 holds in this setting:

LEMMA B.5. *For any database D and* OWALG *query Q not using $\sigma_{\mathbf{Ord}}$, (i) if $Q(D)$ is not ordered then it has one or zero possible worlds where no* Sync *operator has failed. (ii) if $Q(D)$ is ordered then for any possible worlds W and W' of $Q(D)$ where no* Sync *operator has failed, we have $\pi_{-\mathbf{Ord}}(W) = \pi_{-\mathbf{Ord}}(W')$.*

PROOF. We adapt the proof of Lemma B.1 by observing that the case of Sync is analogous to the case of Shuffle (the worlds where Shuffle fails are ignored. Whenever a Shuffle operator fails in all possible worlds, the result has no possible worlds whatsoever, and the conditions are true. □

We now prove Theorem 3.9 by structural induction on the query Q, and by case disjunction on the last operator applied.

For all operations except Sync, we observe that the arguments given in the proof of Theorem 3.6 still apply to general posets. For indeed, the restriction operation and the operation of Lemma B.4 can still be applied to general posets to yield general posets, and all invocations of Lemma B.1 can be replaced with invocations of Lemma B.5. (The possibility that there are no possible worlds where no Sync operator failed is not a problem, as in such cases this will be the case of the whole expression, and this is specifically permitted by the statement of Theorem 3.9.

We thus focus on the case where the last operator is Sync, and the subqueries $Q_1(D)$ and $Q_2(D)$ of Sync are assumed to represent labeled posets $P_1 = (V_1, <_1, \Sigma, \mu_1)$ and $P_2 = (V_2, <_2, \Sigma, \mu_2)$. (By the condition on Sync, the label set of both posets must be the same.) We exclude the empty possible world for Sync as such possible worlds are specifically prohibited by Definition 3.4.

As $Q_1(D)$ and $Q_2(D)$ cannot contain duplicate tuples, the label of each element of P_1 (resp. P_2) is unique among elements of P_1 (resp. P_2), and as the identity of the underlying order elements is arbitrary we can assume that the elements with same labels have the same identity. We consider the labeled poset $P = (V, <, \Sigma, \mu)$ defined on $V = V_1 \cup V_2$ (under this interpretation) by taking μ defined from μ_1 and μ_2 (as we said, $\mu_1(x) = \mu_2(x)$ for all $x \in V_1 \cap V_2$), and defining $<$ in the following way, inspired by the definition of Sync: for all $x, y \in V$, we have $x < y$ if one of $x <_1 y$, $x <_2 y$ holds (with the interpretation that, say, $x <_1 y$ is *undefined* if $x \notin V_1$ or $y \notin V_1$, so it does not hold), and if none of the $y <_1 x$, $y <_2 x$ holds. Enforce transitivity by taking the transitive closure of $<$. (We will deal with irreflexivity later.)

We first argue that this mimics exactly the definition of Sync. If $x < y$, then, letting t_1 and t_2 be the corresponding tuples of S, t_2 cannot precede t_1, for in this case, by our definition, some tuples t_1' and t_2' for t_1 and t_2 are such that t_1' precedes t_2' in one of R, R', and no tuples t_1'', t_2'' for t_1, t_2 can be such that t_2'' precedes t_1'' in the other table. Conversely, if t precedes t' in every possible world of S, then t' cannot precede t in a possible world of S, meaning that one of $x <_1 y$, $x <_2 y$ holds and none of $y <_1 x$, $y <_2 x$ holds. So, by our assumption that Sync succeeds, the existence of a relation satisfying the constraints imposed by the definition of Sync ensures that P is indeed irreflexive, for a cycle in the $<$ relation as defined would mean that the constraints assert that certain tuples must occur in a cycle, which cannot be realized in a possible world of Sync.

Hence, we now know that P captures the possible worlds of $Q(D) = \text{Sync}(Q_1(D), Q_2(D))$: considering the total order induced by a non-empty possible world of $Q(D)$, it suffices to check that it is a linear extension of P, namely, that it is compatible with P, but we have just shown that the order constraints respected by all possible worlds of $Q(D)$ are consistent with P; conversely a linear extension of P must respect the constraints of P, which ensures that it is a possible world of $Q(D)$. Hence, $Q(D)$ really represents the labeled poset P.

Second claim. We now prove the second direction: for any product $\Sigma = D_1 \times \cdots \times D_n$ of domains and poset P on Σ, there exists an OWALG query Q and a database D of size polynomial in P such that $Q(D)$ represents P.

et $P = (V, \Sigma, \mu, <)$ be the labeled poset to represent.

Let $C = \{(x,y) \mid x, y \in V, x < y\}$. For each $(x,y) \in C$, define the ordered relation T_{xy} as follows, with attributes $\mathsf{A1}, \ldots, \mathsf{An}$ with domains respectively $D_1, \ldots, D_n$ and attribute A with domain V, writing $\mu(z) = (\mu_1(z), \ldots, \mu_1(z))$ for all $z \in V$:

A	A1	…	An	O
x	$\mu_1(x)$	…	$\mu_n(x)$	0
y	$\mu_1(y)$	…	$\mu_n(y)$	1

and the ordered relation T_x as follows:

A	A1	…	An	O
x	$\mu_1(x)$	…	$\mu_n(x)$	0

Let T be the $\Pi_{-\mathbf{Ord}}$ of the Sync of the $Rank_{O,<}(T_p)$ for $p \in P$. and $Rank_{O,<}T_x$ for $x \in V$. This expression is of polynomial size in $(V, <)$, as is the input database.

Every Sync must succeed by Proposition 2.12 because there are no conflicts between any of the T_x and T_{xy}, so there are non-empty possible worlds except if P is the empty poset (which is trivial).

Consider a table T' representing a linear extension of $(V, \Sigma, \mu, <)$. By definition of C, it is compatible with all of the order constraints of the T_p's, and thanks to the T_x's its domain is exactly that of the possible words of T. Hence, T' is a possible world of T.

Conversely, consider a non-empty possible world T' of T. Its domain must be V because of the T_x's, and it specifies a total order consistent with P because, if $x < y$ in V, letting t_x and t_y be the tuples of the Sync (before the projection) that correspond to x and y, t_x must have preceded t_y thanks to the constraint imposed by T_{xy}; so T' represents a total order consistent with all constraints of $(V, \Sigma, \mu, <)$, that is, a linear extension of $(V, \Sigma, \mu, <)$.

C. PROOFS FOR SECTION 4

C.1 Proof of Theorem 4.3

NP-hardness. Consider the UNARY-3-PARTITION problem [16]: given $3m$ integers $N = (n_1, \ldots, n_{3m})$ written in unary 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 $\mathcal{I}$ of UNARY-3-PARTITION to an instance $\mathcal{I}'$ of the POSS problem for a query and for relations of the right form.

The domain of relations R and R' will be an attribute A (storing a unique ID) and an attribute L (storing a label):

- The domain of A is v_i^-, v_i^j, v_i^+ for all $1 \leqslant i \leqslant 3m$ and $1 \leqslant j \leqslant n_i$.
- The domain of L is $\{l_-, l, l_+\}$.
- For every $1 \leqslant i \leqslant 3m$, define the ordered relations R_i and R_i':

A	L	Ord
v_i^-	l_-	0
v_i^1	l	1
$\vdots$	$\vdots$	$\vdots$
$v_i^{n_i}$	l	n_i

A	L	Ord
v_i^-	l_-	0
$v_i^{n_i}$	l	1
$\vdots$	$\vdots$	$\vdots$
v_i^1	l	n_i

- The encoding of I is the two relations defined by $R = R_1 \cdots R_{3m}$ and $R' = R'_{3m} \cdots R'_1$, where $\cdot$ denotes concatenation (adjusting the **Ord** values). Note that this is *not* the Concat operator, rather, it is computed offline when performed the reduction (so that R and R' are actual relations of the mandated form).
- The possible world to test is $I = I' \cdots I'$, with m occurrences of I', where I' is as follows:

L	Ord
l_-	0
l_-	1
l_-	2
l	3
$\vdots$	$\vdots$
l	B+2
l_+	B+3
l_+	B+4
l_+	B+5

- The query is $\Pi_{\mathsf{L}}(\mathrm{Sync}(R,R'))$.

Clearly the instance $\mathcal{I}'$ can be computed in polynomial time from $\mathcal{I}$ (remember that the n_i are coded in unary) and $\mathcal{I}'$ respects the desired conditions. In particular, as $\Pi_{-\mathbf{Ord}}(R) = \Pi_{-\mathbf{Ord}}(R')$, $E = \mathrm{Sync}(R,R')$ does not fail by Proposition 2.13. It is now easily observed that, for all $x,y \in \mathrm{dom}(A)$, the only order constraints $x < y$ enforced by E are for cases where x and y are associated to the same n_i and we have either $x = v_i^-$ or $y = v_i^+$. (When they are not associated to the same n_i, R and R' disagree on their order, and for other cases where they are associated to the same n_i, namely where both are of the form v_i^j, then R and R' also disagree about their order.)

We prove that the original UNARY-3-PARTITION instance $\mathcal{I}$ has a solution if and only if the POSS instance $\mathcal{I}'$ has one.

Assume that $\mathcal{I}$ has a solution: $P = P_0, \ldots, P_{m-1}$ where each P_i is a triple of integers of N. We can see the partition P as defining two functions f, g such that, for all i, n_i is the $g(i)$-th element of $P_{f(i)}$ ($0 \leqslant g(i) \leqslant 2$). Consider the possible world where v_i^- appears at position $f(i) \times (B+6) + g(i)$, v_i^+ at position $f(i) \times (B+6) + B + 3 + g(i)$, and v_i^j at position $f(i) \times (B+6) + 3 + j$, for all i and j. It is well-defined (each element of E occurs at exactly one position and the positions are indeed $\{0, \ldots, 3m-1\}$), it is really a possible world (it respects all of the order constraints imposed by E as characterized before) and its projection to L achieves the desired possible world I.

Conversely, from a solution to $\mathcal{I}'$, consider the possible world W such that $proj_L(W) = I$. Consider three consecutive l^- labels of I, and the elements $v_{i_1}^-$, $v_{i_2}^-$ and $v_{i_3}^-$ whose projection achieves these occurrences of l^-. Necessarily the three elements whose projections achieve the three next consecutive occurrences of l^+ must be $v_{i_1}^+$, $v_{i_2}^+$ and $v_{i_3}^+$. (Proof by induction from the first l^-'s and l^+'s to the last ones.) Thus, the elements achieving the B intermediate l elements must be the $v_{i_k}^j$ for $k \in \{1, \ldots, 3\}$, $1 \leqslant j \leqslant i_k$. Thus they correspond a 3-tuple of elements of N whose sum is B, and all those groups of l-elements summing to B yield a solution for $\mathcal{I}$.

Hence the original 3-partition instance has a solution if and only if the POSS problem it encodes to has a solution, and the encoding PTIME, which establishes hardness.

NP membership. To solve the POSS problem in NP, evaluate the query Q on the database D, and, for each nondeterministic subexpression, choose a possible world nondeterministically. As the query is fixed, such choices can be represented in size polynomial in the input database D. Now, having evaluated $Q(D)$ to one of its possible worlds, check if it is the desired possible world I or not. The computation accepts if and only if I is indeed a possible world of $Q(D)$.

Acknowledgement. This proof is inspired by the construction by user Marzio De Biasi[8] to prove the hardness of deciding the existence of bijections between partial orders of dimension 2 (representable as intervals). Our setting differs as one of the orders (the possible world to test) is total, but we can use the non-order attributes as labels to enforce constraints on the bijection.

C.2 Proof of Theorem 4.6

We first detail the proof in the case where $k = 2$.

We write R, R' the arguments of Sync (that we evaluate to full relations, in PTIME data complexity) and I the possible world to test. If I is the empty relation, we can return true, otherwise we do as follows.

We define the function $g(i,j) = |\Pi_{-\mathbf{Ord}}(R[0:i]) \cup \Pi_{-\mathbf{Ord}}(R'[0:j])|$ where $R[0:i]$ denotes the tuples of R at positions from 0 to i (inclusive) in the total order. This function is clearly computable in PTIME.

Define the following aliases, where $\stackrel{?}{=}$ denotes an equality test:

- $F_R(i,j) = \Pi_A(R[i]) \stackrel{?}{=} I[g(i,j)]$
- $F_{R'}(i,j) = \Pi_A(R'[i]) \stackrel{?}{=} I[g(i,j)]$

Define a function $f : \{0, \ldots, |R|\} \times \{0, \ldots, |R'|\} \mapsto \{\text{true}, \text{false}\}$ as follows:

1. $f(|R|, |R'|) = \text{true}$,
2. $f(|R|, j) = F_{R'}(|R|, j) \wedge f(|R|, j+1)$
3. $f(i, |R'|) = F_R(i, |R'|) \stackrel{?}{=} I[g(i, |R'|)]) \wedge f(i+1, |R'|)$,
4. $f(i,j)$, if $\Pi_{-\mathbf{Ord}}(R[i]) \notin \Pi_{-\mathbf{Ord}}(R')$ but $\Pi_{-\mathbf{Ord}}(R'[j]) \in \Pi_{-\mathbf{Ord}}(R)$, is $F_R(i,j) \wedge f(i+1, j)$
5. $f(i,j)$, if $\Pi_{-\mathbf{Ord}}(R'[j]) \notin \Pi_{-\mathbf{Ord}}(R)$ but $\Pi_{-\mathbf{Ord}}(R[i]) \in \Pi_{-\mathbf{Ord}}(R')$, is $F_{R'}(i,j) \wedge f(i, j+1)$
6. $f(i,j)$, if $\Pi_{-\mathbf{Ord}}(R'[j]) \notin \Pi_{-\mathbf{Ord}}(R)$ and $\Pi_{-\mathbf{Ord}}(R[i]) \notin \Pi_{-\mathbf{Ord}}(R')$, is $(F_R(i,j) \wedge f(i+1,j)) \vee (F_{R'}(i,j) \wedge f(i, j+1))$

[8] http://cstheory.stackexchange.com/a/19415

7. $f(i,j)$, if $\Pi_{-\mathbf{Ord}}(R[i]) = \Pi_{-\mathbf{Ord}}(R'[j])$, is $F_R(i,j) \wedge F_{R'}(i,j) \wedge \wedge f(i+1,j+1)$.
8. $f(i,j) =$ false otherwise.

Intuitively, the value of $f(i,j)$ reflects the following: having passed the first i elements of relation R and the first j elements of relation R', which match the first $g(i,j)$ letters of w, can I match the rest of w with the rest of R and R'? The gloss of the rules is the following:

1. If we matched all of I, we have nothing more to do.
2. If we have matched all tuples of R, we must check if all remaining tuples of R' match the rest of I.
3. Likewise symmetrically.
4. If we have one element from $\Pi_{-\mathbf{Ord}}(R) \setminus \Pi_{-\mathbf{Ord}}(R')$ and one element of $\Pi_{-\mathbf{Ord}}(R) \cap \Pi_{-\mathbf{Ord}}(R')$, we cannot pass the one from $\Pi_{-\mathbf{Ord}}(R) \cap \Pi_{-\mathbf{Ord}}(R')$ yet, so we match take the one from $\Pi_{-\mathbf{Ord}}(R) \setminus \Pi_{-\mathbf{Ord}}(R')$ with the current position in I.
5. Likewise symmetrically.
6. If we have one element from $\Pi_{-\mathbf{Ord}}(R) \setminus \Pi_{-\mathbf{Ord}}(R')$ and one from $\Pi_{-\mathbf{Ord}}(R') \setminus \Pi_{-\mathbf{Ord}}(R)$, we can take choose to match either of those two (this is the only case where we have a choice).
7. If the same element is at the current position in both $\Pi_{-\mathbf{Ord}}(R)$ and $\Pi_{-\mathbf{Ord}}(R')$, we must take it from both relations and check that it matches.
8. Otherwise, we do nothing.

Computing these values bottom-up (from $f(|R|,|R'|)$ to $f(0,0)$) can be done in PTIME and it is easily seen that f matches its informal meaning.

For larger values of k, we generalize this method to a function f of n parameters with the same intuition: at each step, we can take an element from the current position at any sequence as long as all of its occurrences are at the current position of their respective sequences, and we must enforce that the label of this element is the correct one, and pop it from all the sequences where it appears; the running time is polynomial if the number of sequences is fixed.

As for the fact that Sync operators may fail, observe that, whenever some Sync operator fails, failures occurring in its arguments are irrelevant. So, to take into account possible failures of the Sync operators, we can apply the above algorithm first assuming that no Sync fails. If it returns false, we can assume that the innermost Sync fails, replacing it with an empty relation, and applying the algorithm again. If it fails again, we can assume that the second innermost Sync fails, etc., until we reach the outermost Sync. So this only adds a multiplicative factor of k, which we assume is fixed.

C.3 Proof of Theorem 4.7

We know from [24] that the following problem (MIN-SAT) is NP-complete:

Input A set $U = \{x_1, \ldots, x_k\}$ of k variables, a collection $C = \{c_1, \ldots, c_n\}$ of n clauses over U such that each clause $c \in C$ has $|c| = 2$ literals, a positive integer $d \leqslant n$.

Output Whether there exists an assignment of U such that no more than d clauses in C are true.

We now describe how to encode an instance $\mathcal{I}$ of MIN-SAT to an instance $\mathcal{I}'$ of POSS. The domain of the input relations will be A and L, where A is a unique ID and L is a label.

- $\mathrm{dom}(L)$ is $\{l_1, \ldots, l_k\} \cup \{l_C\}$.
- $\mathrm{dom}(A)$ is $\bigcup_{1 \leqslant i \leqslant x} \{e_i^+, e_i^-\} \cup \{e_j^C \mid 1 \leqslant j \leqslant n\}$.
- The encoding of a clause $c_j = \pm_1 x_p \vee \pm_2 x_q$ is two ordered relations R_j^1 and R_j^2 defined as follows:

A	L	**Ord**
$e_p^{\pm_1}$	l_p	0
e_j^C	l_C	1

A	L	**Ord**
$e_q^{\pm_2}$	l_q	0
e_j^C	l_C	1

- I is $\Pi_{\mathsf{L},\mathbf{Ord}}(I')$ where I' is the following relation (with attribute P added for convenience later):

L	**Ord**	P
l_1	0	p_1^l
$\vdots$	$\vdots$	$\vdots$
l_k	$k-1$	p_k^l
l_C	k	p_1^C
$\vdots$	$\vdots$	$\vdots$
l_C	$k+n-d-1$	p_{n-d}^C
l_1	$k+n-d$	p_1^r
$\vdots$	$\vdots$	$\vdots$
l_k	$2k+n-d$	p_k^r
l_C	$2k+n-d+1$	p_1^C
$\vdots$	$\vdots$	$\vdots$
l_C	$2k+n$	p_n^C

- The query is $\pi_{\mathsf{L}}(\mathrm{Sync}(\mathcal{R}))$ where $\mathcal{R} = \bigcup_{1 \leqslant i \leqslant n} \{R_i^1, R_i^2\}$, where we abusively write Sync as an n-ary operator to mean that we nest binary invocations of Sync; so the query is $(2n)$-safe and has no SPJ subqueries.

Clearly the instance $\mathcal{I}'$ can be computed in polynomial time from the original instance $\mathcal{I}$.

By Proposition 2.12, as there are no conflicts between any of the tables, every Sync succeeds. We prove that $\mathcal{I}$ has a solution if and only if $\mathcal{I}'$ has one.

First, assume that $\mathcal{I}$ has a solution: let $\nu : U \to \{-,+\}$ be the valuation of the x_i's such that $\leqslant d$ clauses of C are true under ν. Let $C_{i_1}, \ldots, C_{i_n}$ be a renumbering of the clauses so that $C_{i_1}, \ldots, C_{i_{n-d}}$ are false.

We consider the possible world W defined by the following function f indicating the row of I' with which we want to match every element of $\bigcup_{j,a} \Pi_{-\mathbf{Ord}}(R_j^a)$: $f(e_i^{\pm_1}) = p_i^r$ if $\nu(x_i) = \pm_1$ and p_i^l if $\nu(x_i) \neq b$, and $f(e_{i_j}^C) = p_j^C$. Clearly this is a bijection from $\mathrm{dom}(A)$ to $\mathrm{dom}(P)$, so we must only prove that this is a possible world, namely, that it respects all the order constraints of the tables $\mathcal{R}$.

To prove that it is indeed a possible world, consider some relation of $\mathcal{R}$:

A	L	Ord
$e_i^{\pm_1}$	l_i	0
e_j^C	l_C	1

and show that the one corresponding order constraint (informally, $e_i^{\pm_1} < e_j^C$) is preserved. If the clause c_j is true under ν, then $f(e_j^C) = p_{j'}^d$ for some $d \in \{l,r\}$ and $j' > n-d$, so that $f(e_j^C) > p_i^{b'}$ for any b' and i, so indeed $f(e_j^C) > f(e_i^b)$. Otherwise, if c_j is false, we must have $\nu(x_i) \neq b$, because otherwise the occurrence of x_i or $\neg x_i$ in c_j (depending on whether $b = +$ or $b = -$) would make c_j true. Hence, $f(e_i^b) = p_i^l$, and indeed $p_i^l < p_j^C$ for all j so $f(e_i^b) < f(e_j^C)$. So f really defines a possible world which achieves I, so $\mathcal{I}'$ has a solution.

Conversely, assume that $\mathcal{I}'$ has a solution that we represent by a bijection $f : \mathrm{dom}(A) \to \mathrm{dom}(P)$ as before. We must have $f(e_i^b) \in \{p_i^l, p_i^r\}$, so we can define an assignment ν on U by: $\nu(x_i) = +$ if $f(e_i^+) = p_i^r$ and $\nu(x_i) = -$ if $f(e_i^+) = p_i^l$. Observe that whenever $f(e_i^+) = p_i^l$ we must have $f(e_i^-) = p_i^r$ because f is bijective (observation (*)).

We now show that ν makes at least $n-d$ clauses false. Specifically, we consider the clauses $c_{j_1}, \ldots, c_{j_{n-d}}$ such that the image by f of the corresponding $e_{j_1}^C, \ldots, e_{j_{n-d}}^C$ are the p_j^C with $j \leqslant n-d$, and show that those $n-d$ clauses must be false under ν. Consider such a clause c_q. Let x_i be a positive literal occurring in c. By construction the following relation occurs in a Sync subexpression:

A	L	Ord
e_i^+	l_i	0
e_q^C	l_C	1

Hence we must have $f(e_i^+) < f(e_q^C)$ because f is order-preserving. Because $f(e_q^C) = p_j^C$ with $j \leqslant n-d$, and $f(e_i^+) \in \{p_i^l, p_i^r\}$ because $proj_A(W) = w$, and $p_j^C < p_i^r$, we must have $f(e_i^+) = p_i^l$ so by definition of ν we must have $\nu(x_i) = -$, so the occurrence of x_i in c_j does not make c_j true. Symmetrically, if $\neg x_i$ is a positive literal in c, we must have $f(e_i^-) = p_i^l$, so by observation (*) $f(e_i^+) = p_i^r$ and $\nu(x_i) = +$ so the occurrence of $\neg x_i$ in c_j does not make c_j true. Hence, ν witnesses that $\mathcal{I}$ has a solution.

We have proved that $\mathcal{I}$ has a solution if and only if $\mathcal{I}'$ has one, and the encoding is clearly PTIME. Hence, the NP-hardness follows from that of MIN-SAT.

Acknowledgement. This proof is inspired by the construction by user domotorp[9] to prove the hardness of deciding the existence of topological sorts satisfying a compatibility condition between vertices and positions. In our setting, the order on which to perform the topological sort is encoded as applications of Sync, and the compatibility is enforced using the non-Ord attributes as labels.

[9] `http://cstheory.stackexchange.com/a/19081`

D. PROOFS FOR SECTION 5

D.1 Proof of Proposition 5.1

Consider the relation $R = \{(0,s),(1,t)\}$ whose only attribute is *Id* and *Ann*, and the OWALG query $R' = \mathrm{Rank}_{Id,<_{\mathbb{N}}}(R)$. Assume that R' can be represented as an (M,K)-relation. By the semantics of the Rank operator, $\{(0,s,0)\}$ and $\{(1,t,0)\}$ are possible worlds of R'. Hence, by monotonicity of (M,K)-relations, there must be a possible world of R' containing both tuples $\{(0,s,0),(1,t,0)\}$, but it is not well-formed, contradicting the intended semantics of Rank.

D.2 Proof of Proposition 5.8

Let $Q \in \text{OWALG}'$ be a query. We note that by definition of OWALG$'$, $Q = \mathrm{op}(Q')$ where Q' is a positive relational algebra query. Let D be a $\mathbb{B}$-database, then $Q(D) = \mathrm{op}(Q'(D))$. We know from [20] that $Q'(D) = D'$ is a $\mathbb{B}$-database and Q' is set-compatible. As noted, set-compatibility for the deterministic case is indeed a particular case of our set-compatibility definition.

So it is enough to show that the single application of op satisfies set-compatibility. This is shown through a by-case reasoning on op:

Rank. The Rank operation is deterministic, thus its provenance representation includes no element of X and we only need to show that $Rank(supp(R)) = supp(\iota(Rank(R))$ for every $\mathbb{B}$-relation R. First, note that up to the *Ord* attribute, $Rank(R)$ has the same tuples as R with the same annotations. So in particular their subsets of tuples annotated with $\top$ coincide up to the *Ord* attribute. This means that up to the *Ord* attribute, we have even $Rank(supp(R)) = supp(Rank(R))$. To see that the corresponding values in the *Ord* attribute coincide after applying ι, observe that for each tuple $s \in supp(Rank(R))$ (whose corresponding tuple in $supp(R)$ is t) we have

$$s.\mathbf{Ord} = \sum_{\{t' \in supp(R) | t'.A < t.A\}} 1_{\mathbb{B}} \otimes 1_{\mathbb{N}} + \sum_{\{t' \notin supp(R) | t'.A < t.A\}} 0_{\mathbb{B}} \otimes 1_{\mathbb{N}}$$

$$= 1_{\mathbb{B}} \otimes \sum_{\{t' \in supp(R) | t'.A < t.A\}} 1_{\mathbb{N}}$$

Where the last transformation is enabled due to the axioms imposed on the structure. By applying ι we get

$$\sum_{\{t' \in supp(R) | t'.A < t.A\}} 1_{\mathbb{N}}$$

which is exactly

$$|\{t' \in supp(R) \mid t'.A < t.A\}|$$

i.e. the rank of t in $supp(R)$.

Concat. Again, Concat is deterministic so we need to show $\mathrm{Concat}(supp(R), supp(R')) = supp(\iota(\mathrm{Concat}(R,R'))$ for any $\mathbb{B}$-relations R, R'. Here again, we first show that the tuples of the two sides of the equation coincide up to the *Ord* attribute. Note that a tuple in $\mathrm{Concat}(R,R')$ is annotated with the annotation of its counterpart in R or R' (recall that if a tuple appears in both, then two "copies" of it up to the Ord attribute will be generated in the result of Concat). So a tuple is in

the support of the Concat result if and only if its counterpart in R or R' is in the input. As for the *Ord* attribute, note that for a tuple s such that $s = h(r)$ (referring to the h used in the definition of Concat) for $r \in R$ then for the boolean case we get

$s.\mathbf{Ord} = \sum_{\{r_1 \in R | r_1.\mathbf{Ord} < r.\mathbf{Ord}\}} (R(r_1) \otimes 1_{\mathbb{N}})$
$= \sum_{\{r_1 \in supp(R) | r_1.\mathbf{Ord} < r.\mathbf{Ord}\}} 1_{\mathbb{B}} \otimes 1_{\mathbb{N}}$
$= 1_{\mathbb{B}} \otimes \sum_{\{r_1 \in supp(R) | r_1.\mathbf{Ord} < r.\mathbf{Ord}\}} 1_{\mathbb{N}}$
And by applying ι we get $\sum_{\{r_1 \in supp(R) | r_1.\mathbf{Ord} < r.\mathbf{Ord}\}} 1_{\mathbb{N}}$
$= |\{r_1 \in supp(R) \mid r_1.\mathbf{Ord} < r.\mathbf{Ord}\}|$.

For $s = I(r')$ we get via the same transformations that $s.Ord = |supp(R)| + |\{r_1 \in supp(R') \mid r_1.\mathbf{Ord} < r.\mathbf{Ord}\}|$. This is exactly the Ord value of s in the (not annotated) relation $\text{Concat}(supp(R), supp(R'))$.

Shuffle. The Shuffle operator is nondeterministic, and so uses variables in X. The same arguments of compatibility up to the Ord attribute given for Rank and Concat continue to hold for Shuffle (note that up to the *Ord* attribute the values are the same in all possible worlds). We then show set-compatibility by induction on the number of tuples that are annotated with $\bot$. We show that every valuation yields exactly the set of correct worlds with consecutive ordering.

Induction basis. We need to show that if all tuples are annotated with $\top$, then the possible valuations to $Y = \{Y_0, Y_1, X_2, ..., Y_n\}$ yield exactly the possible valuations to the additional **Ord** attribute that are "correct" according to the nondeterministic semantics of Shuffle when applied on the un-annotated R, R'. For the first direction, consider a given order consistent with the orders in R, R'. Let Y_0 be the number of elements from R' that precede R_0 in the order, and for $i > 0$ let Y_i be number of elements from R' that are between R_{i-1} and R_i. Note that this uniquely defines the order. Now the order we obtain for s_i in the construction is $i + | \{j \leqslant m \mid j < \sum_{k \leqslant i} Y_k\}$. The former is exactly the number of elements of R that precede R_i, while the latter is by definition of Y_k exactly the number of elements of R' that precede R_i. Similarly for s'_i we get $i + | \{j \mid \sum_{m \leqslant j} Y_m \leqslant i\}$. Now the former is exactly the number of elements of R that precede R_i, while the latter is exactly the number of elements of R' whose "offset" is less or equal to i, i.e. precede R_i. For the converse we note that *any* valuation to the Y_i's encodes an order in the above way (note that since we are using each Y_i as a relative offset, there is no need to impose any constraints on the assigned values) ; the only subtlety is in the case $V(\sum_{j \leqslant i} Y_i) > m$ for some i – but note that such valuation yields the same possible world as a valuation that "truncates" any value greater than m to be m (in both cases it means all tuples of R' will appear exactly before R_i in a consecutive order.

Induction step. We assume correctness for N $\bot$ values and prove for $N+1$. Let $r_j \in R$ be a tuple whose annotation "flips" from $\top$ to $\bot$, and note that (1) If $i < j$ then $s_i.\mathbf{Ord}$ remains unaffected (with respect to the world in which r_j was annotated with $\top$), (2) if $i > j$ then $s_i.\mathbf{Ord}$ is lower by 1, (3) if for a given valuation and given i, r_j was chosen to precede r'_i (i.e. ($\left[\sum_{m \leqslant j} Y_m < i\right)$ then the order of r'_i is lower by 1 and (4) otherwise the order of r'_i stays intact. This means that the "flip" of annotation has the correct affect on any order (and any valuation).

Sync. The same arguments of compatibility up to the Ord attribute given for Rank and Concat continue to hold for Shuffle (note that up to the *Ord* attribute, and up to the empty relation which is a possible world in both sides of the equation, the values are the same in all possible worlds). For values in the *Ord* attribute we consider the possible valuations to the $NewOrd(t)$ variables and note that for any given valuation we obtain consecutive values in the *Ord* attribute respecting the order imposed by the valuation (i.e. the i'th tuple according to its *NewOrd* value will have *Ord* value i). If the valuation satisfies the ordering constraints imposed by shuffle, then we obtain a possible world of $\text{Sync}(supp(R), supp(R'))$. Otherwise we get the empty relation, which is also a possible world.

D.3 Proof of Proposition 5.9

Similarly to the proof of set-compatibility (Section D.2), it is enough to consider the order-aware operators, and show commutation with homomorphisms for it (as commutation with homomorphisms for the positive relational algebra operators was shown in [20]). In all of the following let K, K' be commutative semirings and let D be a K-database.

The application of h on the annotated result of an order-aware query has two affects: the first is mapping every tuple annotation a to $h(a)$. Here note that by definition $h(a_1 + a_2) = h(a_1) + h(a_2)$ and $h(a_1 \cdot a_2) = h(a_1) \cdot h(a_2)$, so that commutation holds. The second is mapping values in Ord attributes. Here we use our definition of lifting h to $K(X)$: the effect of the lifted operation is to replace each occurrence of k by k'. This means that we can invariably apply h on annotations of D to obtain a K'-database and then apply an order-aware operator, or first apply the operator and then the mapping. In both cases we get get equivalent expressions in $K(X)$.

D.4 Proof of Proposition 5.11

Poly-size overhead of each of the SPJ operators was shown in [20]. For each of the newly defined operators:

Rank *and* Concat. The *Ord* expression for each tuple is linear in the number of tuples (involves sum over all tuples), so the result is of overall quadratic size.

Shuffle. The *Ord* expression for each tuple is at most of quadratic size in the number of tuples, so the result is of overall cubic size.

Sync. The dominant factor here is G which is of quadratic size in the number of tuples, and appears as the annotation of every tuple, so the result is of overall cubic size.

And so we have poly-size overhead in data complexity.

D.5 Proof of Theorem 5.13

We next explain the required changes to the semantics, to obtain a semantics for the full OWALG language.

Rank In defining the Rank operator, we have assumed an order relation on the A attribute according to which ranking is performed. If values in the A attribute (We

assume for simplicity that ranking is done based on a single attribute) are in fact elements in $(K(X)_{nest} \otimes \mathbb{N}$ (e.g. one uses the Rank operator with the Ord attribute as parameter, after applying shuffle), it may be the case that the order between them cannot be decided. To this end we replace in the definition:

$$t.ord = \sum_{\{t' \in supp(R) | t'.A < t.A\}} R(t') \otimes 1_M$$

with

$$t.ord = \sum_{\{t' \in supp(R)\}} (R(t') \cdot [t'.A < t.A]) \otimes 1_M$$

Concat Similarly, in defining the Concat operator, instead of enumerating tuples satisfying $r_1.\mathbf{Ord} < r.\mathbf{Ord}$ we enumerate over all tuples and multiply by an abstract condition $[r_1.\mathbf{Ord} < r.\mathbf{Ord}]$. Namely $\sum_{\{r_1 \in R | r_1.\mathbf{Ord} < r.\mathbf{Ord}\}} (R(r_1) \otimes 1_{\mathbb{N}})$ is replaced by $\sum_{r_1 \in supp(R)} ([r_1.\mathbf{Ord} < r.\mathbf{Ord}] \cdot (R(r_1)) \otimes 1_{\mathbb{N}}$

Shuffle In defining the Shuffle operator, we have assumed that we can know which is the i'th tuple; instead, we choose some arbitrary order over the tuples and impose the order using condition, utilizing the fact that to check if a tuple t is in location k we can use $[\sum_{\{t' \in supp(R)\}} (R(t') \cdot [t'.Ord < t.Ord]) \otimes 1_M = k]$. And so we can define:

$$\begin{aligned} s_i.\mathbf{Ord} = & \sum_{j<n} (R(r_j) \cdot [R(r_j).Ord < R(r_i).Ord]) \otimes 1_{\mathbb{N}}) \\ & + \sum_{loc<n} \Big[\sum_{\{t' \in supp(R)\}} (R(t') \cdot [t'.Ord < R(r_i).Ord]) \\ & \otimes 1_M = loc \Big] \\ & \cdot \sum_{j \leqslant m} \Big(\Big(R'(r_j) \cdot \Big[j < \sum_{k \leqslant n} Y_k \cdot [k < loc] \Big] \Big) \otimes 1_{\mathbb{N}} \Big) \end{aligned}$$

And symmetrically apply the changes for the definition of $s'_i.\mathbf{Ord}$.

$$\begin{aligned} s'_i.\mathbf{Ord} = & \sum_{j<n} (R'(r_j) \cdot [R'(r'_j).Ord < R'(r'_i).Ord]) \otimes 1_{\mathbb{N}}) \\ & + \sum_{loc<n} \Big[\sum_{\{t' \in supp(R)\}} \\ & (R(t') \cdot [t'.Ord < R(r_i).Ord]) \otimes 1_M = loc \Big] \\ & \cdot \\ & \sum_{j \leqslant n} \Big(\Big(\Big[\sum_{m \leqslant j} Y_m \leqslant loc \Big] \cdot R(r_j) \Big) \otimes 1 \Big) \end{aligned}$$

Sync For Sync we need to encode the precedes relation, used in the construction in the global condition G, using expressions. For a tuple t in the sync result let $r(t)$ be its origin from R and $r'(t)$ be its origin from R'. For each t, t' we then use the following "macro": $notprecedes(t,t')$ is encoded by:

$$\begin{aligned} & [[r(t) \otimes 1 = 0] \otimes 1 + [r(t') \otimes 1 = 0] \otimes 1 \\ & + [r'(t) \otimes 1 = 0] \otimes 1 + [r'(t') \otimes 1 = 0] \otimes 1 \\ & + ([r(t).Ord < r(t').Ord] \cdot [r'(t).Ord < r'(t').Ord]) \otimes 1 \\ & = 0] \end{aligned}$$

And then:

$$\begin{aligned} G = \Pi_{\{t,t' \in R \cup R'\}} & [notprecedes(t,t') + [Y_t < Y_t] \cdot \\ & \Pi_{\{t \neq t' \in R \cup R'\}} [Y_t \neq Y_{t'}] \neq 0] \end{aligned}$$

Observe that the obtained expressions are in $K(X)_{nest}$ (comparison expressions involve tensors with $\mathbb{N}$ of elements from K or other equation elements (which by definition then have a lower level of nesting). We can then adapt the proofs of Propositions 5.8, 5.9 and 5.11. For set-compatibility, we then note that when the semiring is $\mathbb{B}$ the equation elements encode the same conditions that were encoded as part of the construction for the non-nested case. Since the defined notion of applying a valuation over just nested expressions in this case corresponds to testing the equation and replacing it with $\bot$ or $\top$ according to its truth values, this means that the same arguments for set-compatibility of the individual operators apply. The proof then proceeds by induction on the query structure. The inductive step for SPJ is immediate, and that for the order-aware operators was given above. We further note that the queries combine sub-expression in a way that commutes with homomorphisms, either through the semiring structure (and then it applies by definition of a homomorphism) or though the generation and combination of (nested) equation elements or tensors, for which again semiring homomorphism commutes. For poly-size overhead we note that the changes in construction, while cumbersome to write, did not increase the asymptotic dependency of the output size on the input database size.