Sequenced Semantics for Temporal Multiset Relations (Extended Version)

Anton Dignös
Free University of Bozen-Bolzano
dignoes@inf.unibz.it

Boris Glavic
Illinois Institute of Technology
bglavic@iit.edu

Xing Niu
Illinois Institute of Technology
xniu7@hawk.iit.edu

Michael Böhlen
University of Zurich
boehlen@ifi.uzh.ch

Johann Gamper
Free University of Bozen-Bolzano
gamper@inf.unibz.it

ABSTRACT

Sequenced semantics is widely used for evaluating queries over temporal data: temporal relations are seen as sequences of snapshot relations, and queries are evaluated at each snapshot (aka *snapshot-reducibility*). For efficiency and usability, sequenced semantics is typically implemented using interval-based temporal data models. In this paper we show that, maybe surprisingly, all state-of-the-art methods using intervals fail to be snapshot-reducible. We identify two common bugs: Sequenced aggregation does not report values at time points where no data is present (*aggregation gap bug*), whereas sequenced difference fails to return a result for time points in multiset relations if its left input contains more duplicates of a tuple than the right input (*bag difference bug*). Moreover, the interval-encoding of temporal data is typically not unique. We tackle these problems and propose the first provably correct interval-based framework for sequenced semantics over temporal set *and* multiset relations which supports full relational algebra plus aggregation. We start by defining a general conceptual model to lay out the semantic foundation of our framework. Our core contribution is a new logical model that represents temporal relations as K-relations, where each tuple is annotated with its complete history. The logical model (i) is equivalent to the conceptual model, (ii) determines a unique encoding for the implementation using the SQL standard's tuple-timestamped representation of temporal data, and (iii) induces a correct rewriting scheme for queries over this encoding. We implement our approach as a database middleware and demonstrate experimentally that, even without specialized operator implementations, the performance of our approach is competitive with native implementations of temporal operators.

PVLDB Reference Format:
. . *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/TBD

1. INTRODUCTION

works

name	skill	period
Ann	SP	$[03, 10)$
Joe	NS	$[08, 16)$
Sam	SP	$[08, 16)$
Ann	SP	$[18, 20)$

assign

mach	skill	period
M1	SP	$[03, 12)$
M2	SP	$[06, 14)$
M3	NS	$[03, 16)$

(a) Input period relations

Q_{onduty}

cnt	period
0	$[00, 03)$
1	$[03, 08)$
2	$[08, 10)$
1	$[10, 16)$
0	$[16, 18)$
1	$[18, 20)$
0	$[20, 24)$

(b) Sequenced aggregation

$Q_{skillreq}$

skill	period
SP	$[06, 08)$
SP	$[10, 12)$
NS	$[03, 08)$

(c) Sequenced difference

Figure 1: Sequenced semantics query evaluation – highlighted tuples are erroneously omitted by approaches that exhibit the aggregation gap (AG) and bag difference (BD) bugs.

Temporal databases have been studied for decades. Recently, there is renewed interest in the topic fueled by the fact that abundant storage has made long term archival of historical data feasible. This has led to the incorporation of temporal features into the SQL:2011 standard [27] and to their implementation in database management systems (DBMSs), e.g., PostgreSQL [34], Teradata [42], Oracle [30], IBM DB2 [35], and MS SQLServer [29]. However, none of these systems, with the partial exception of Teradata, supports *sequenced semantics* [6], an important class of temporal queries. Given a temporal database, a query Q under sequenced semantics returns a temporal relation that assigns to each point in time the result of evaluating Q over the snapshot of the database at this point in time. This fundamental correctness property of sequenced semantics is known as *snapshot-reducibility* [28, 40]. The SQL:2011 standard defines an encoding of temporal data where each tuple is associated with a validity period. We refer to such relations as *SQL period relations* in this work. Note that SQL period relations use multiset semantics, i.e., multiple duplicates of a tuple may exist at the same point in time.

Example 1.1 (Sequenced Aggregation)**.** *Consider the SQL period relation* `works` *in Figure 1a that records factory workers, their skills, and the hours when they are on duty. The validity period of each tuple is stored in the temporal attribute* `period`*. The time domain in this example spans the full hours of one day, from* 00*, which corresponds to 12:00am (midnight), to* 24 *(exclusive), which corresponds to 12:00am of the next day. For instance, the first tu-*

ple records that specialized worker Ann with skill SP is on duty between 3:00am (03) and 10:00am (10). The company has a safety policy requiring that at least one specialized worker is in the factory at any given time. This can be checked using the following aggregation query under sequenced semantics.

Q_{onduty}:
```
SELECT count(*) AS cnt FROM works
WHERE skill = 'SP'
```

Evaluated under sequenced semantics, this query returns the number of specialized workers that are on duty at any given point of time. The result is shown in Figure 1b. For instance, at 08:00am two specialized workers (Ann and Joe) are on duty. The query exposes several safety violations, e.g., no specialized worker is on duty between 12:00am and 3:00am.

In the example above, safety violations correspond to gaps, i.e., periods of time where the aggregation's input is empty. As we will demonstrate, all approaches with sequenced semantics that we are aware of do not return aggregation results for gaps (tuples marked in red) and, therefore, violate snapshot-reducibility. As a consequence, in our example they fail to identify safety violations. We refer to this type of error as the *aggregation gap bug* (*AG bug*).

Similar to the case of aggregation, we also identify a common error related to sequenced bag difference (**EXCEPT ALL**).

Example 1.2 (Sequenced Bag Difference)**.** *Consider again Figure 1. Relation* `assign` *records machines (mach) that need to be assigned to workers with a specific skill over a specific period of time. For instance, the third tuple records that machine M3 requires a non-specialized (NS) worker for the time period* $[03, 16)$. *To determine which skill sets are missing during which time period, we evaluate the following query under sequenced semantics:*

$Q_{skillreq}$:
```
SELECT skill FROM assign
EXCEPT ALL
SELECT skill FROM works
```

The result in Figure 1c indicates that one more specialized worker (SP) is required during the periods $[06, 08)$ *and* $[10, 12)$ *and one more non-specialized worker (NS) for the period* $[03, 08)$.

The problem here is that many approaches treat bag difference as a **NOT EXISTS** subquery, and therefore do not return a tuple t from the left input relation if this tuple exists in the right input (independent of their multiplicity). For instance, the two tuples for the specialized worker (highlighted in red) are not returned, since there exists an SP worker at each snapshot in the `works` relation. This violates snapshot-reducibility. We refer to this type of error as the *bag difference bug* (*BD bug*).

The interval-based representation of temporal relations creates an additional problem: the encoding of a temporal query result is typically not unique. For instance, tuple $(Ann, SP, [03, 10))$ from the `works` relation in Figure 1 can equivalently be represented as two tuples $(Ann, SP, [03, 08))$ and $(Ann, SP, [08, 10))$. We refer to a method that determines how temporal data and sequenced query results are grouped into intervals as an *interval-based representation system*. A unique and predictable representation of temporal data is a desirable property for various reasons. First, if a sequenced query result is used as input to a non-sequenced query that accesses the time dimension (e.g., the period attribute of an SQL period relation), the result of this query may depend on the particular representation [43]. Second, equivalent relational algebra expressions should not lead to syntactically different result relations. These problems can be addressed by using a representation system that associates a unique encoding with each temporal database. Furthermore, overlap between multiple periods associated with a tuple and unnecessary splits of periods complicate the interpretation of data and, thus, should be avoided if possible.

In this paper, we address the above limitations of previous approaches for sequenced semantics and develop a framework based on the following desiderata: (i) support for set and multiset relations, (ii) snapshot-reducibility for all operations, and (iii) a unique interval-based encoding of temporal relations. We address these desiderata using a three-level approach. First, we introduce a *conceptual model* that supports both sets and multisets, and by definition is snapshot-reducible. This model, however, uses a verbose encoding of temporal data and, thus, is not practical. In fact, its sole purpose is to provide a yardstick for correctness. Afterwards, we develop a more compact *logical model* as a representation system, where the complete temporal history of a tuple is stored in an annotation attached to the tuple. The conceptual and the logical models leverage the theory of K-relations, which are a general class of annotated relations that cover both set relations and multiset relations. For our *implementation*, we use standard SQL over period relations to ensure compatibility with the SQL:2011 standard and existing DBMSs. We prove the equivalence between the three layers (i.e., the conceptual model, the logical model and the implementation) and show that the logical model determines a unique interval-encoding for the implementation and a correct rewriting scheme for queries over this interval-encoding.

Our main technical contributions can be summarized as follows:

- We introduce *sequenced K-relations* as a generalization of sequenced set and multiset relations. These relations are by definition snapshot-reducible and, thus, are a perfect choice for a *conceptual model*.
- We define an interval-based representation, termed *period K-relations*, as our *logical model* and prove that these relations are a compact and unique representation system for sequenced semantics over sequenced K-relations. We show this for the full relational algebra plus aggregation ($\mathcal{RA}^{agg}$).
- We achieve the unique encoding of temporal data as period K-relations through K-coalescing, a generalization of set based coalescing [9].
- We demonstrate that the multiset version of period K-relations can be encoded as *SQL period relations*, a common interval-based model in DBMSs, and how to translate queries with sequenced semantics over period K-relations into standard SQL queries over this encoding.
- We implement our approach as a database middleware and present optimizations that eliminate redundant coalescing steps. Our experimental comparison with other approaches shows that we do not need to sacrifice performance to achieve correctness.

The remainder of this paper is organized as follows. In Section 2, we review related work and discuss which approaches exhibit the AG or BD bugs. Afterwards, we present an overview of our approach in Section 3. In Section 4, we introduce sequenced K-relations. Section 5 covers temporal K-elements, the annotations of our logical model, and a coalescing-based normal form for these elements. We introduce period K-relations in Section 6 and study the evaluation of complex queries (difference and aggregation) over period K-relations in Section 7. We then study how period K-relations can be encoded as SQL period relations (i.e., multiset relations with a period attribute) and present a translation of queries with sequenced semantics into standard multiset semantics over this encoding (Section 8). Section 9 describes an implementation of our approach, which is experimentally evaluated in Section 10. We conclude in Section 11.

2. RELATED WORK

Temporal Query Languages. There is a long history of research on temporal query languages [23, 5]. Many temporal query languages including TSQL2 [37, 38], ATSQL2 (Applied TSQL2) [7], IXSQL [28], ATSQL [8], and SQL/TP [44] support sequenced semantics. In this paper, we do not focus on a particular query language. Rather, we provide a general framework that can be used to correctly implement sequenced semantics over period set and multiset relations for any language.

Interval-based Approaches for Sequenced Semantics. In the following, we discuss interval-based approaches for sequenced semantics. Table 1 shows for each approach whether it supports multisets, whether it is free of the aggregate gap and bag difference bugs, and whether its interval-based encoding of a sequenced query result is unique. An N/A indicates that the approach does not support the operation for which this type of bug can occur or the semantics of this operation is not defined precisely enough to judge its correctness. Note that while temporal query languages may be defined to apply sequenced semantics and, thus, by definition are snapshot-reducible, (the specification of) their implementation might fail to be snapshot-reducible. In the following discussion of the temporal query languages in Table 1, we refer to their semantics as provided in the referenced publication(s).

Interval preservation (ATSQL) [8, Def. 2.10] is a representation system for SQL period relations (multisets) that tries to preserve the intervals associated with input tuples, i.e., fragments of all intervals (including duplicates) associated with the input tuples "survive" in the output. Interval preservation is snapshot-reducible for multiset semantics for positive relational algebra, but exhibits the aggregate gap and bag difference bug. Moreover, the period encoding of a query result is not unique as it depends both on the query and the input representation. *Teradata* [42] is a commercial DBMS that supports sequenced operators using ATSQL's statement modifiers. The implementation is based on query rewriting [1] and does not support difference. Teradata's implementation exhibits the aggregate gap bug. Since the application of coalescing is optional, the encoding of snapshot relations as period relations is not unique. *Change preservation* [17, Def. 3.4] determines the interval boundaries of a query result tuple t based on the intervals attached to tuples in t's provenance. Whether such an encoding of a result is unique for equivalent queries depends on the provenance model that is employed. If the provenance model is insensitive to query rewrite [13], i.e., equivalent queries have the same provenance, then the corresponding change-preservation encoding is unique for a given interval-timestamped input relation (but not necessarily for different snapshot-equivalent inputs). Change preservation was originally studied for set semantics employing the lineage provenance model in [15] and the PI-CS model in [17] (which for positive queries is equivalent to provenance polynomials in the semiring annotation framework [19]). However, none of the two models has the same equivalences as set semantics (the correct choice would be semiring PosBool(X) [21, 20] aka Minimal Why-provenance [13]). *TSQL2* [37, 38, 40] implicitly applies coalescing [9] to produce a unique representation. Thus, it supports only set semantics, and it does not support aggregation. Snodgrass et al. [39] present a valid-time extension of *SQL/Temporal* and an algebra with sequenced semantics. The algebra supports multisets, but exhibits both the aggregate gap and bag difference bug. The algebra preserves intervals from the input where possible and, thus, the interval representation of a snapshot relation is not unique. *TimeDB* [41] is an implementation of ATSQL2 [7]. It uses a semantics for bag difference and intersection that is not snapshot-reducible (see [41, pp. 63]).

Table 1: Interval-based approaches for sequenced semantics.

Approach	**Multisets**	**AG bug free**	**BD bug free**	**Unique encoding**
Interval preservation [8] (ATSQL)	✓	×	×	×
Teradata [42]	✓	×	N/A	×[1]
Change preservation [15, 17]	×	×	N/A	×
TSQL2 [37, 38, 40]	×	N/A	N/A	✓
ATSQL2 [7]	✓	N/A	×	×
TimeDB [41] (ATSQL2)	✓	N/A	×	×
SQL/Temporal [39]	✓	×	×	×
Our approach	✓	✓	✓	✓

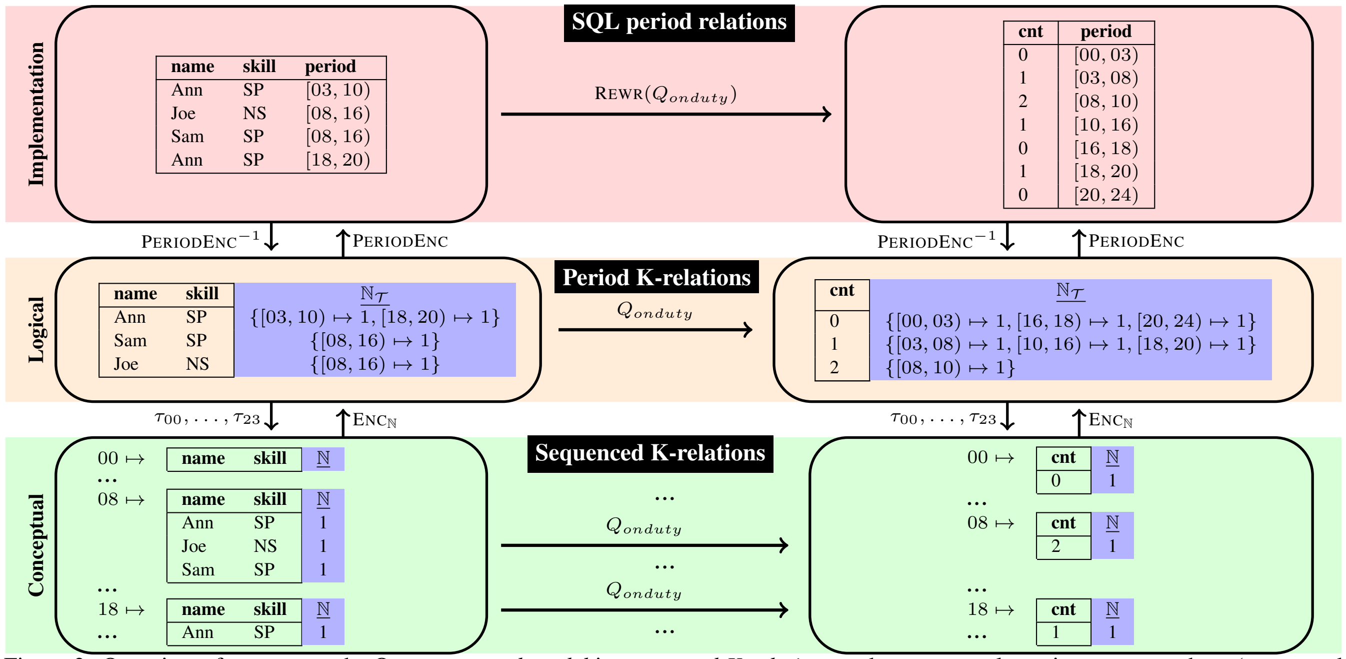
Our approach is the first that supports set and multiset relations, is resilient against the two bugs, and specifies a unique interval-encoding for the implementation.

Non-sequenced Temporal Queries. Non-sequenced temporal query languages, such as IXSQL [28] and SQL/TP [44], do not explicitly support sequenced semantics. Nevertheless, we review these languages here since they allow to express queries with sequenced semantics. IXSQL [28] introduces several interval-specific conditions (e.g., checking for overlap) and two operations, fold and unfold, that translate between point-based and interval-based representations of temporal data. The language essentially treats temporal attributes as regular data. To write a temporal query, the user has to combine (un)fold, regular non-temporal algebra operators, and the new temporal conditional constructs. SQL/TP [44] introduces a point-wise semantics for temporal queries [11, 43]. Time is handled as a regular attribute with the exception that relations can be infinite in the time dimension. The language supports multiset semantics. Intervals are used as an efficient encoding of time points, and a normalization operation is used to split intervals. The interval-based encoding of the language is not unique, since time points are grouped into intervals based on query syntax and based on how the input is encoded as period relations. While this has no effect on the semantics since SQL/TP queries cannot distinguish between different interval-based encodings of a temporal database, it might be confusing to users that observe different query results for equivalent queries/inputs.

Implementations of Temporal Operators. A large body of work has focused on the implementation of individual temporal algebra operators such as joins [16, 32, 10] and aggregation [4, 33, 31]. Some exceptions supporting multiple operators are [25, 17, 12]. These approaches introduce efficient evaluation algorithms for a particular semantics of a temporal algebra operator. Our approach can utilize efficient operator implementations as long as (i) their semantics is compatible with our interval-based encoding of sequenced query results and (ii) they are snapshot-reducible.

Coalescing. Coalescing produces a unique representation of a *set* semantics temporal database. Böhlen et al. [9] study optimizations for coalescing and present algebraic rewrites that eliminate unnecessary coalescing operations. Zhou et al. [45] use SQL:2003 analytical functions to efficiently implement coalescing in SQL. We

[1]Optionally, coalescing (`NORMALIZE ON` in Teradata) can be applied to get a unique encoding at the cost of loosing multiplicities.



name	skill	period
Ann	SP	$[03, 10)$
Joe	NS	$[08, 16)$
Sam	SP	$[08, 16)$
Ann	SP	$[18, 20)$

cnt	period
0	$[00, 03)$
1	$[03, 08)$
2	$[08, 10)$
1	$[10, 16)$
0	$[16, 18)$
1	$[18, 20)$
0	$[20, 24)$

name	skill	$\mathbb{N}_{\mathcal{T}}$
Ann	SP	$\{[03, 10) \mapsto 1, [18, 20) \mapsto 1\}$
Sam	SP	$\{[08, 16) \mapsto 1\}$
Joe	NS	$\{[08, 16) \mapsto 1\}$

cnt	$\mathbb{N}_{\mathcal{T}}$
0	$\{[00, 03) \mapsto 1, [16, 18) \mapsto 1, [20, 24) \mapsto 1\}$
1	$\{[03, 08) \mapsto 1, [10, 16) \mapsto 1, [18, 20) \mapsto 1\}$
2	$\{[08, 10) \mapsto 1\}$

$00 \mapsto$

name	skill	$\mathbb{N}$

…

$08 \mapsto$

name	skill	$\mathbb{N}$
Ann	SP	1
Joe	NS	1
Sam	SP	1

…

$18 \mapsto$

name	skill	$\mathbb{N}$
Ann	SP	1

…

$00 \mapsto$

cnt	$\mathbb{N}$
0	1

…

$08 \mapsto$

cnt	$\mathbb{N}$
2	1

…

$18 \mapsto$

cnt	$\mathbb{N}$
1	1

…

Figure 2: Overview of our approach. Our *conceptual model* is *sequenced K-relations* and nontemporal queries over snapshots (sequenced semantics). Our *logical* model is *period K-relations* and temporal queries corresponding to sequenced queries in the conceptual model. For our *implementation* we use *SQL period relations* and rewritten non-temporal queries implementing the sequenced queries of the two other models. Each model is associated with transformations to the models below and above. As we will demonstrate, all of these transformations commute with queries (modulo the rewriting $\textsc{Rewr}$ when translating from the logical model to the implementation).

generalize coalescing to K-relations and apply the resulting operation, which we call K-coalescing, to define a unique encoding of interval-based temporal relations, including *multiset* relations. Similar to [9], we remove unnecessary K-coalescing steps and, similar to [45], we use OLAP functions to efficiently implement K-coalescing.

Temporality in Annotated Databases. Kostiley et al. [26] is to the best of our knowledge the only previous approach that uses semiring annotations to express temporality. The authors define a semiring whose elements are sets of time points. This approach is limited to set semantics, and no efficient interval-based encoding was presented. The LIVE system [14] combines provenance and uncertainty annotations with versioning. The system uses interval timestamps, and query semantics is based on snapshot-reducibility [14, Def. 2]. However, computing the intervals associated with a query result requires provenance to be maintained for every query result.

3. SOLUTION OVERVIEW

In this section, we give an overview of our three-level framework, which is illustrated in Figure 2.

Conceptual model – Sequenced K-relations. As a *conceptual model* we use sequenced relations which map time points to snapshots. Queries over such relations are evaluated over each snapshot, which trivially satisfies snapshot-reducibility. To support both sets and multisets, we introduce *sequenced K-relations* [21], which are sequenced relations where each snapshot is a K-relation. In a K-relation, each tuple is annotated with an element from a domain K that forms a commutative semiring. For example, relations annotated with elements from the semiring $\mathbb{N}$ (natural numbers) correspond to multiset semantics and relations annotated with elements from the boolean semiring $\mathbb{B}$ to set semantics. The result of a sequenced query Q over a sequenced K-relation is the result of evaluating Q over the K-relation at each time point.

Example 3.1 (Conceptual Model)**.** *Figure 2 (bottom) shows the snapshots at times* `00`*,* `08`*, and* `18` *of an encoding of the running example as sequenced $\mathbb{N}$-relations. Each snapshot is an $\mathbb{N}$-relation where tuples are annotated with their multiplicity (shown on the right of each tuple with shaded background). For instance, the snapshot at time* `08` *has three tuples, each with multiplicity 1. The result of query Q_{onduty} is shown on the bottom right. Every snapshot in the result is computed by running Q_{onduty} over the corresponding snapshot in the input. For instance, at time* `08` *there are two specialized workers, i.e., cnt $= 2$.*

Logical Model – Period K-relations. We introduce period K-relations as a *logical model*. Period K-relations are a compact encoding of temporal data that generalizes period relations. In a *period K-relation*, every tuple is annotated with a *temporal K-element* which is a unique interval-based representation of the tuple's complete history. This avoids splitting a tuple's history across multiple tuples. We define a class of semirings called *period semirings* whose elements are temporal K-elements. Specifically, for any semiring K we can construct a period semiring $K_{\mathcal{T}}$ whose annotations are temporal K-elements. For instance, $\mathbb{N}_{\mathcal{T}}$ is the period semiring corresponding to semiring $\mathbb{N}$ (multisets). Furthermore, we present a bijective transformation $\textsc{Enc}_K$ that constructs a period K-relation from a sequenced K-relation. We define necessary conditions for an interval-based model to correctly encode sequenced K-relations and prove that period K-relations fullfil these conditions. Specifically, we call an interval-based model a *representation system* iff (i) encoding for R is unique, (ii) queries over encodings are snapshot-reducible, and (iii) the encoding of every sequenced K-relation R is snapshot-equivalent to R.

Example 3.2 (Logical Model)**.** *Figure 2 (middle) shows an encoding of the running example as period K-relation. Each tuple is annotated with its complete history, represented as pairs of time intervals and the tuple's multiplicity (an annotation from semiring $\mathbb{N}$) during such time intervals. The annotations are elements of the period semiring $\mathbb{N}_{\mathcal{T}}$. For instance, the result tuple of query Q_{onduty} with $cnt = 0$ has multiplicity 1 during the time intervals $[00, 03)$, $[16, 18)$, and $[20, 24)$. This encoding is produced by applying the transformation $\text{ENC}_{\mathbb{N}}$ to the sequenced $\mathbb{N}$-relation shown in Figure 2. The snapshot at time T can be restored by applying the time slice operator τ_T which we will introduce in Section 4.2.*

Implementation – SQL Period Relations. To ensure compatibility with the SQL:2011 standard, we use standard period relations (also denoted as *SQL period relations*) in our *implementation* and translate sequenced semantics queries into standard SQL queries over these period relations. For this we define an encoding of $\mathbb{N}_{\mathcal{T}}$-relations as SQL period relations (PERIODENC) together with a rewriting scheme for queries (REWR).

Example 3.3 (Implementation)**.** *Consider the SQL period relations shown on the top of Figure 2. Every interval-annotation pair of a temporal $\mathbb{N}$-element in the logical model is encoded as a separate tuple in the implementation. For instance, the tuple (Ann, SP) in the logical model is encoded in the implementation as two tuples, each of which records one of the two intervals during which the tuple exists.*

We present an implementation of our framework as a database middleware that exposes sequenced temporal queries as a new language feature in SQL and rewrites sequenced queries into standard SQL queries over SQL period relations. That is, we can directly evaluate sequenced queries over data stored as native period relations. Since many temporal query languages support sequenced semantics, our results enable the efficient and correct implementation of sequenced queries specified in these languages.

4. SEQUENCED K-RELATIONS

We first review background on the semiring annotation framework (K-relations). Afterwards, we define *sequence K-relations* as our conceptual model and sequenced semantics for this model. Importantly, queries over sequenced K-relations are snapshot-reducible by construction. Finally, we state requirements for a logical model to be a representation system for this conceptual model.

4.1 K-relations

In a K-relation [21], every tuple is annotated with an element from a domain K of annotations. By selecting an appropriate domain K, this framework can be tailored for a wide variety of purposes. For instance, to model multiset semantics we annotate each tuple with its multiplicity (i.e., an element from the natural numbers $\mathbb{N}$). K-relations also support many extensions of the relational model including incomplete databases, trust, and provenance.

The semiring framework requires the domain K to be a commutative semiring. A structure $(K, +_K, \cdot_K, 0_K, 1_K)$ over a set K where addition $+_K : K \times K \to K$ and multiplication $\cdot_K : K \times K \to K$ are binary operations over K is a commutative semiring iff (i) the addition and multiplication operations are commutative, associative, and have a neutral element (0_K and 1_K, respectively); (ii) multiplication distributes over addition; and (iii) multiplication with zero returns zero. Abusing notation, we will use K to denote both a semiring structure as well as its domain.

The semiring addition and multiplication operations are used to define how annotations propagate through queries. The fact that K is a commutative semiring guarantees that expected algebraic equivalences hold over K-relations. Consider a universal countable domain $\mathcal{U}$ of values. Formally, an n-ary K-relation R over $\mathcal{U}$ is a (total) function that maps tuples (elements from $\mathcal{U}^n$) to elements from K with the convention that tuples mapped to 0_K are not in the relation. Furthermore, we require that $R(t) \neq 0_K$ only holds for finitely many t. Two semirings are of particular interest to us: The semiring $(\mathbb{B}, \vee, \wedge, false, true)$ with elements *true* and *false* using $\vee$ as addition and $\wedge$ as multiplication corresponds to set semantics. The semiring $(\mathbb{N}, +, \cdot, 0, 1)$ of natural numbers with standard arithmetics corresponds to multiset semantics.

4.1.1 Queries over K-relations

The operators of the positive relational algebra ($\mathcal{RA}^+$) over K-relations are defined by applying the $+_K$ and $\cdot_K$ operations of the semiring K to input annotations. Intuitively, the $+_K$ and $\cdot_K$ operations of the semiring correspond to the alternative and conjunctive use of tuples, respectively. For instance, if an output tuple t is produced by joining two input tuples annotated with k and k', then the tuple t is annotated with $k \cdot_K k'$. Below we provide the standard definition of $\mathcal{RA}^+$ over K-relations [21]. For a tuple t, we use $t.A$ to denote the projection of t on a list of projection expressions A and $t[R]$ to denote the projection of t on the attributes of relation R. For a condition θ and tuple t, $\theta(t)$ denotes a function that returns 1_K if $t \models \theta$ and 0_K otherwise.

Definition 4.1 ($\mathcal{RA}^+$ over K-relations)**.** *Let K be a semiring, R, S denote K-relations, t, u denote tuples of appropriate arity, and $k \in K$. The positive relational algebra $\mathcal{RA}^+$ on K-relations is defined as:*

$$\begin{aligned} \sigma_\theta(R)(t) &= R(t) \cdot \theta(t) && (\textit{selection}) \\ \Pi_A(R)(t) &= \sum\nolimits_{u: u.A = t} R(u) && (\textit{projection}) \\ (R \bowtie S)(t) &= R(t[R]) \cdot S(t[S]) && (\textit{join}) \\ (R \cup S)(t) &= R(t) + S(t) && (\textit{union}) \end{aligned}$$

In the following, we will make use of homomorphisms, functions from the domain of a semiring K_1 to the domain of a semiring K_2 that are well-behaved, i.e., commute with the semiring operations. Since $\mathcal{RA}^+$ over K-relations is defined in terms of these operations, it follows that semiring homomorphisms commute with queries, as was proven in [21].

Definition 4.2 (Homomorphism)**.** *A mapping $h : K_1 \to K_2$ from semiring K_1 to semiring K_2 is called a homomorphism iff for all $k, k' \in K_1$:*

$$\begin{aligned} h(0_{K_1}) &= 0_{K_2} & h(1_{K_1}) &= 1_{K_2} \\ h(k +_{K_1} k') &= h(k) +_{K_2} h(k') & h(k \cdot_{K_1} k') &= h(k) \cdot_{K_2} h(k') \end{aligned}$$

Example 4.1. *Consider the $\mathbb{N}$-relations shown below which are non-temporal versions of our running example. Query $Q = \Pi_{mach}(\,works \bowtie assign)$ returns machines for which there are workers with the right skill to operate the machine. Under multiset semantics we expect M1 to occur in the result of Q with multiplicity 8 since $(M1, SP)$ joins with $(Pete, SP)$ and with (Bob, SP). Evaluating the query in $\mathbb{N}$ yields the expected result by multiplying the annotations of these join partners. Given the $\mathbb{N}$ result of the query, we can compute the result of the query under set semantics by applying a homomorphism h which maps all non-zero annotations to true and 0 to false. For example, for result $(M1)$ we get $h(8) = true$, i.e., this tuple is in the result under set semantics.*

works

name	skill	$\mathbb{N}$
Pete	SP	1
Bob	SP	1
Alice	NS	1

assign

mach	skill	$\mathbb{N}$
M1	SP	4
M2	NS	5

Result

A	$\mathbb{N}$
M1	$1 \cdot 4 + 1 \cdot 4 = 8$
M2	$5 \cdot 1 = 5$

4.2 Sequenced K-relations

We now formally define sequenced K-relations, sequenced semantics over such relations, and then define representation systems. We assume a totally ordered and finite domain $\mathbb{T}$ of time points and use $\leq_{\mathbb{T}}$ to denote its order. T_{min} and T_{max} denote the minimal and maximal (exclusive) time point in $\mathbb{T}$ according to $\leq_{\mathbb{T}}$, respectively. We use $T + 1$ to denote the successor of time point $T \in \mathbb{T}$ according to the order $\leq_{\mathbb{T}}$.

A sequenced K-relation over a relation schema $\mathbf{R}$ is a function $\mathbb{T} \rightarrow \mathcal{R}_{K,\mathbf{R}}$, where $\mathcal{R}_{K,\mathbf{R}}$ is the set of all K-relations with schema $\mathbf{R}$. Sequenced K-databases are defined analog. We use $\mathcal{DB}_{\mathbb{T},K}$ to denote the set of all sequenced K-databases for time domain $\mathbb{T}$.

Definition 4.3 (Sequenced K-relation)**.** *Let K be a commutative semiring and $\mathbf{R}$ a relation schema. A sequenced K-relation R is a function $R : \mathbb{T} \rightarrow \mathcal{R}_{K,\mathbf{R}}$.*

For instance, a sequenced $\mathbb{N}$-relation is shown in Figure 2 (bottom). Given a sequenced K-relation, we use the *timeslice operator* [24] to access its state (snapshot) at a time point T:

Definition 4.4 (Timeslice operator)**.** *Let R be a sequenced K-relation and $T \in \mathbb{T}$. The timeslice operator $\tau_T(R)$ is defined as:*

$$\tau_T(R) = R(T)$$

The evaluation of a query Q over a sequenced database (set of sequenced relations) D under *sequenced semantics* returns a sequenced relation $Q(D)$ that is constructed as follows: for each time point $T \in \mathbb{T}$ we have $Q(D)(T) = Q(D(T))$. Thus, sequenced temporal queries over sequenced K-relations behave like queries over K-relations for each snapshot, i.e., their semantics is uniquely determined by the semantics of queries over K-relations.

Definition 4.5 (Sequenced Semantics)**.** *Let D be a sequenced K-database and Q be a query. The result $Q(D)$ of Q over D is a sequenced K-relation that is defined point-wise as follows:*

$$\forall T \in \mathbb{T} : Q(D)(T) = Q(\tau_T(D))$$

For example, consider the sequenced $\mathbb{N}$-relation shown at the bottom of Figure 2 and the evaluation of Q_{onduty} under sequenced semantics as also shown in this figure. Observe how the query result is computed by evaluating Q_{onduty} over each snapshot individually using multiset ($\mathbb{N}$) query semantics. Furthermore, since $\tau_T(Q(R)) = Q(R)(T)$, per the above definition, the timeslice operator commutes with queries: $\tau_T(Q(R)) = Q(\tau_T(R))$. This property is *snapshot-reducibility*.

4.3 Representation Systems

Sequenced K-relations lead to a clean semantics for sequenced queries, but are impractical due to a high level of redundancy. We will investigate more compact, interval-based representations of temporal information that have a sequenced query semantics that is equivalent with sequenced databases. Compact encodings of temporal databases have been considered in the past. For instance, in [43] point-based representations which are equivalent to our sequenced $\mathbb{B}$-databases have been called abstract temporal databases, while interval-based encodings have been called concrete temporal databases.

We study representation systems that consist of a set of representations $\mathcal{E}$, a function $\textsc{Enc} : \mathcal{E} \rightarrow \mathcal{DB}_{\mathbb{T},K}$ which associates an encoding in $\mathcal{E}$ with the sequenced K-database it represents, and a timeslice operator τ_T which extracts the snapshot at time T from an encoding. If $\textsc{Enc}$ is injective, then we use $\textsc{Enc}^{-1}(D)$ to denote the unique encoding associated with D. We use τ to denote the timeslice over both sequenced databases and representations. It will be clear from the input which operator τ refers to.

For such a representation system, we consider two encodings D_1 and D_2 from $\mathcal{E}$ to be *snapshot-equivalent* [22] (written as $D_1 \sim D_2$) if they encode the same sequenced K-database. Note that this is the case if they encode the same snapshots, i.e., iff for all $T \in \mathbb{T}$ we have $\tau_T(D_1) = \tau_T(D_2)$. For a representation system to behave correctly, the following conditions have to be met: 1) **uniqueness**: for each snapshot K-database D there exists a unique element from $\mathcal{E}$ representing D; 2) **snapshot-reducibility**: the timeslice operator commutes with queries; and 3) **snapshot-preservation**: the encoding function $\textsc{Enc}$ preserves the snapshots of the input.

Definition 4.6 (Representation System)**.** *We call a triple $(\mathcal{E}, \textsc{Enc}, \tau)$ a* representation system *for sequenced semantics queries over sequenced K-databases with regard to a class of queries $\mathcal{C}$ iff for every sequenced database D, encodings E, $E' \in \mathcal{E}$, time point T, and query $Q \in \mathcal{C}$ we have*

1. $\textsc{Enc}(E) = \textsc{Enc}(E') \Rightarrow E = E'$ *(uniqness)*
2. $\tau_T(Q(E)) = Q(\tau_T(E))$ *(snapshot-reducibility)*
3. $\textsc{Enc}(E) = D \Rightarrow \tau_T(E) = \tau_T(D)$ *(snapshot-preservation)*

5. TEMPORAL K-ELEMENTS

We now introduce temporal K-elements that are the annotations we use to define our logical model which we will later prove to be a representation system for sequenced K-relations. Temporal K-elements record, using an interval-based encoding, how the K-annotation of a tuple in a sequenced K-relation changes over time. We introduce a unique normal form for temporal K-elements based on a generalization of coalescing [9].

5.1 Defining Temporal K-elements

To define temporal K-elements, we need to introduce some background on intervals. Given the time domain $\mathbb{T}$ and its associated total order $\leq_{\mathbb{T}}$, an interval $I = [T_b, T_e)$ is a pair of time points from $\mathbb{T}$, where $T_b <_{\mathbb{T}} T_e$. Interval I represents the set of contiguous time points $\{T \mid T \in \mathbb{T} \wedge T_b \leq_{\mathbb{T}} T <_{\mathbb{T}} T_e\}$. For an interval $I = [T_b, T_e)$ we use I^+ to denote T_b and I^- to denote T_e. We use $I, I', I_1, \ldots$ to represent intervals. We define a relation $adj(I_1, I_2)$ that contains all interval pairs that are adjacent: $adj(I_1, I_2) \Leftrightarrow (I_1{}^- = I_2{}^+) \vee (I_2{}^- = I_1{}^+)$. We will implicitly understand set operations, such as $t \in I$ or $I_1 \subseteq I_2$, to be interpreted over the set of points represented by an interval. Furthermore, $I \cap I'$ denotes the interval that covers precisely the intersection of the sets of time points defined by I and I' and $I \cup I'$ denotes their union (only well-defined if $I \cap I' \neq \emptyset$ or $adj(I, I')$). For convenience, we define $I \cup I' = \emptyset$ iff $I \cap I' = \emptyset \wedge \neg adj(I, I')$. We use $\mathbb{I}$ to denote the set of all intervals over $\mathbb{T}$.

Definition 5.1 (Temporal K-elements)**.** *Given a semiring K, a* temporal K-element $\mathcal{T}$ *is a function $\mathbb{I} \rightarrow K$. We use $\mathbb{TE}_K$ to denote the set of all such temporal elements for K.*

Temporal K-elements are represented as sets of input-output pairs. All intervals that are not explicitly mentioned are assumed to be mapped to 0_K.

Example 5.1. *Reconsider the running example and assume that $\mathbb{T} = \{00, \dots, 23\}$ (we are considering closed intervals now). The history of the annotation of tuple $t =$ `(Ann, SP)` from the `works` relation is as shown in Figure 2 (middle). For sake of the example, we change the multiplicity of this tuple to 3 during $[03, 09)$ and 2 during $[18, 20)$. This information is encoded in the temporal $\mathbb{N}$-element shown below:*

$$\mathcal{T}_1 = \{[03, 09) \mapsto 3, [18, 20) \mapsto 2\}$$

Note that a temporal K-element $\mathcal{T}$ may map overlapping intervals to non-zero elements of K. In principle, we could disallow this, but allowing overlap simplifies the definition of operations over temporal K-elements. We assign the following semantics to overlap: the annotation at a time point T recorded by $\mathcal{T}$ is the sum of the annotations assigned to intervals containing T. For instance, the annotation at time 04 for the $\mathbb{N}$-element $\mathcal{T} = \{[00, 05) \mapsto 2, [04, 05) \mapsto 1\}$ would be $2 + 1 = 3$. To extract the annotation valid at time T from a temporal K-element $\mathcal{T}$, we define a timeslice operator for temporal K-elements as follows:

$$\tau_T(\mathcal{T}) = \sum_{T \in I} \mathcal{T}(I) \qquad \text{(timeslice operator)}$$

Given two temporal K-elements $\mathcal{T}_1$ and $\mathcal{T}_2$, we would like to know if they represent the same history of annotations. For that, we define *snapshot-equivalence* ($\sim$) for temporal K-elements:

$$\mathcal{T}_1 \sim \mathcal{T}_2 \Leftrightarrow \forall T \in \mathbb{T} : \tau_T(\mathcal{T}_1) = \tau_T(\mathcal{T}_2) \quad \text{(snapshot-equivalence)}$$

5.2 A Normalform Based on K-Coalescing

The encoding of the annotation history of a tuple as a temporal K-element is typically not unique.

Example 5.2. *Reconsider the temporal $\mathbb{N}$-element $\mathcal{T}_1$ from Example 5.1. Recall that intervals not shown are mapped to 0. The $\mathbb{N}$-elements shown below are snapshot-equivalent to $\mathcal{T}_1$.*

$$\mathcal{T}_2 = \{[03, 09) \mapsto 1, [03, 06) \mapsto 2, [06, 09) \mapsto 2, [18, 19) \mapsto 2\}$$

$$\mathcal{T}_3 = \{[03, 05) \mapsto 3, [05, 09) \mapsto 3, [18, 19) \mapsto 2\}$$

To be able to build a representation system based on temporal K-elements we need a unique way to encode the annotation history of a tuple as a temporal K-element (condition 1 of Definition 4.6). That is, we need to define a normal form for temporal K-elements such that snapshot-equivalent temporal elements are equal if brought into this normal form. To this end, we generalize *coalescing*, which was defined for temporal databases with set semantics in [36, 9]. The generalized form, which we call K-coalescing, coincides with standard coalescing for semiring $\mathbb{B}$ (set semantics) and, for any semiring K, yields a unique encoding of snapshot-equivalent temporal K-elements.

K-coalescing creates maximal intervals of contiguous time points with the same annotation. The output is a temporal K-element such that (a) no two intervals mapped to a non-zero element overlap and (b) adjacent intervals assigned to non-zero elements are guaranteed to be mapped to different annotations. To determine such intervals, we define annotation changepoints, time points T where the annotation of a temporal K-element differs from the annotation at $T - 1$, i.e., $\tau_T(\mathcal{T}) \neq \tau_{T-1}(\mathcal{T})$). It will be convenient to also consider T_{min} as an annotation changepoint.

Definition 5.2 (Annotation Changepoint). *Given a temporal K-element $\mathcal{T}$, a time point T is called a* changepoint *in $\mathcal{T}$ if one of the following conditions holds:*

- $T = T_{min}$ *(smallest time point)*
- $\tau_{T-1}(\mathcal{T}) \neq \tau_T(\mathcal{T})$ *(change of annotation)*

sal	period
50k	$[1, 13)$
30k	$[3, 13)$
30k	$[3, 10)$
40k	$[11, 13)$

$$\mathcal{T}_{50k} = \{[1, 13) \mapsto 1\}$$
$$\mathcal{T}_{30k} = \{[3, 10) \mapsto 1, [3, 13) \mapsto 1\}$$
$$\mathcal{T}_{40k} = \{[11, 13) \mapsto 1\}$$

Figure 3: Example period multiset relation S and temporal $\mathbb{N}$-elements encoding the history of tuples.

We use $CP(\mathcal{T})$ to denote the set of all annotation changepoints for $\mathcal{T}$. Furthermore, we define $CPI(\mathcal{T})$ to be the set of all intervals that consist of consecutive change points:

$$\begin{aligned} CPI(\mathcal{T}) = \{[T_b, T_e) \mid\ & T_b <_{\mathbb{T}} T_e \wedge T_b \in CP(\mathcal{T}) \wedge \\ & (T_e \in CP(\mathcal{T}) \vee T_e = T_{max}) \wedge \\ & \nexists T' \in CP(\mathcal{T}) : T_b <_{\mathbb{T}} T' <_{\mathbb{T}} T_e\} \end{aligned}$$

In Definition 5.2, $CPI(\mathcal{T})$ computes maximal intervals such that the annotation assigned by $\mathcal{T}$ to each point in such an interval is constant. In the coalesced representation of $\mathcal{T}$ only such intervals are mapped to non-zero annotations.

Definition 5.3 (K-Coalesce). *Let $\mathcal{T}$ be a temporal K-element. We define K-coalescing $\mathcal{C}_K$ as a function $\mathbb{TE}_K \to \mathbb{TE}_K$:*

$$\mathcal{C}_K(\mathcal{T})(I) = \begin{cases} \tau_{I^+}(\mathcal{T}) & \text{if } I \in CPI(\mathcal{T}) \\ 0_K & \text{otherwise} \end{cases}$$

We use $\mathbb{TEC}_K$ to denote all normalized temporal K-elements, i.e., elements $\mathcal{T}$ for which $\mathcal{C}_K(\mathcal{T}) = \mathcal{T}'$ for some $\mathcal{T}'$.

Example 5.3. *Consider the SQL period relation shown in Figure 3. The three temporal $\mathbb{N}$-elements encode the history of the three tuples $(30k)$, $(40k)$ and $(50k)$. Note that $\mathcal{T}_{30k}$ is not coalesced since the two non-zero intervals of this temporal $\mathbb{N}$-element overlap. Applying $\mathbb{N}$-coalesce we get:*

$$\mathcal{C}_{\mathbb{N}}(\mathcal{T}_{30k}) = \{[3, 10) \mapsto 2, [10, 13) \mapsto 1\}$$

That is, this tuple occurs twice within the time interval $[3, 10)$ and once in $[10, 13)$, i.e., it has annotation changepoints 3, 10, and 14. Interpreting the same relation under set semantics (semiring $\mathbb{B}$), the history of $(30k)$ can be encoded as a temporal $\mathbb{B}$-element $\mathcal{T}_{30k}{}' = \{[3, 10) \mapsto true, [3, 13) \mapsto true\}$. Applying $\mathbb{B}$-coalesce we get:

$$\mathcal{C}_{\mathbb{B}}(\mathcal{T}_{30k}{}') = \{[3, 13) \mapsto true\}$$

That is, this tuple occurs (is annotated with $true$) within the time interval $[3, 13)$ and its annotation changepoints are 3 and 14.

We now prove several important properties of the K-coalesce operator establishing that $\mathbb{TEC}_K$ (coalesced temporal K-elements) is a good choice for a normalform of temporal K-elements.

Lemma 5.1. *Let K be a semiring and $\mathcal{T}$, $\mathcal{T}_1$ and $\mathcal{T}_2$ temporal K-elements. We have:*

$$\mathcal{C}_K(\mathcal{C}_K(\mathcal{T})) = \mathcal{C}_K(\mathcal{T}) \qquad \text{(idempotence)}$$
$$\mathcal{T}_1 \sim \mathcal{T}_2 \Leftrightarrow \mathcal{C}_K(\mathcal{T}_1) = \mathcal{C}_K(\mathcal{T}_2) \qquad \text{(uniqueness)}$$
$$\mathcal{T} \sim \mathcal{C}_K(\mathcal{T}) \qquad \text{(equivalence preservation)}$$

Proof. All proofs are shown in Appendix A. □

6. PERIOD SEMIRINGS

Having established a unique normal form of temporal K-elements, we now proceed to define period semirings as our logical model. The elements of a period semiring are temporal K-elements in normal form. We prove that these structures are semirings and ultimately that relations annotated with period semirings are representation systems for sequenced K-relations for $\mathcal{RA}^+$. In Section 7, we then prove them to also be a representation system for $\mathcal{RA}^{agg}$, i.e., queries involving difference and aggregation.

When defining the addition and multiplication operations and their neutral elements in the semiring structure of temporal K-elements, we have to ensure that these definitions are compatible with semiring K on snapshots. Furthermore, we need to ensure that the output of these operations is guaranteed to be K-coalesced. The latter can be ensured by applying K-coalesce to the output of the operation. For addition, snapshot reducibility is achieved by pointwise addition (denoted as $+_{K_\mathcal{P}}$) of the two functions that constitute the two input temporal K-elements. That is, for each interval I, the function that is the result of the addition of temporal K-elements $\mathcal{T}_1$ and $\mathcal{T}_2$ assigns to I the value $\mathcal{T}_1(I) +_K \mathcal{T}_2(I)$. For multiplication, the multiplication of two K-elements assigned to an overlapping pair of intervals I_1 and I_2 is valid during the intersection of I_1 and I_2. Since both input temporal K-elements may assign non-zero values to multiple intervals that have the same overlap, the resulting K-value at a point T would be the sum over all pairs of overlapping intervals. We denote this operation as $\cdot_{K_\mathcal{P}}$. Since $+_{K_\mathcal{P}}$ and $\cdot_{K_\mathcal{P}}$ may return a temporal K-element that is not coalesced, we define the operations of our structures to apply $\mathcal{C}_K$ to the result of $+_{K_\mathcal{P}}$ and $\cdot_{K_\mathcal{P}}$. The zero element of the temporal extension of K is the temporal K-element that maps all intervals to 0 and the 1 element is the temporal element that maps every interval to 0_K except for $[T_{min}, T_{max})$ which is mapped to 1_K.

Definition 6.1 (Period Semiring). *For a time domain $\mathbb{T}$ with minimum T_{min} and maximum T_{max} and a semiring K, the period semiring $K_\mathcal{T}$ is defined as:*

$$K_\mathcal{T} = (\mathbb{TEC}_K, +_{K_\mathcal{T}}, \cdot_{K_\mathcal{T}}, 0_{K_\mathcal{T}}, 1_{K_\mathcal{T}})$$

where for $k, k' \in \mathbb{TEC}_K$:

$$\forall I \in \mathbb{I} : 0_{K_\mathcal{T}}(I) = 0_K$$

$$\forall I \in \mathbb{I} : 1_{K_\mathcal{T}}(I) = \begin{cases} 1_K & \textit{if } I = [T_{min}, T_{max}) \\ 0_K & \textit{otherwise} \end{cases}$$

$$k +_{K_\mathcal{T}} k' = \mathcal{C}_K(k +_{K_\mathcal{P}} k')$$

$$\forall I \in \mathbb{I} : (k +_{K_\mathcal{P}} k')(I) = k(I) +_K k'(I)$$

$$k \cdot_{K_\mathcal{T}} k' = \mathcal{C}_K(k \cdot_{K_\mathcal{P}} k')$$

$$\forall I \in \mathbb{I} : (k \cdot_{K_\mathcal{P}} k')(I) = \sum_{\forall I', I'' : I = I' \cap I''} k(I') \cdot_K k'(I'')$$

Example 6.1. *Consider the $\mathbb{N}_\mathcal{T}$-relation `works` shown in Figure 2 (middle) and query $\Pi_{skill}(works)$. Recall that the annotation of a tuple t in the result of a projection over a K-relation is the sum of all input tuples which are projected onto t. For result tuple `(SP)` we have input tuples `(Ann, SP)` and `(Sam, SP)` with $\mathcal{T}_1 = \{[03,10) \mapsto 1, [18,20) \mapsto 1\}$ and $\mathcal{T}_2 = \{[08,16) \mapsto 1\}$, respectively. The tuple `(SP)` is annotated with the sum of these annotations, i.e., $\mathcal{T}_1 +_{\mathbb{N}_\mathcal{T}} \mathcal{T}_2$. Substituting definitions we get:*

$$\mathcal{T}_1 +_{\mathbb{N}_\mathcal{T}} \mathcal{T}_2 = \mathcal{C}_\mathbb{N}(\mathcal{T}_1 +_{\mathbb{N}_\mathcal{P}} \mathcal{T}_2)$$
$$= \mathcal{C}_\mathbb{N}(\{[03,10) \mapsto 1, [18,20) \mapsto 1, [08,16) \mapsto 1\})$$
$$= \{[03,08) \mapsto 1, [08,10) \mapsto 2, [10,16) \mapsto 1, [18,20) \mapsto 1\}$$

*Thus, as expected, the result records that, e.g., there are two skilled workers (*SP*) on duty during time interval* $[08, 10)$.

Having defined the family of period semirings, it remains to be shown that $K_\mathcal{T}$ with standard K-relational query semantics is a representation system for sequenced K-relations.

6.1 $K_\mathcal{T}$ is a Semiring

As a first step, we prove that for any semiring K, the structure $K_\mathcal{T}$ is also a semiring. The following lemma shows that K-coalesce can be redundantly pushed into $+_{K_\mathcal{P}}$ and $\cdot_{K_\mathcal{P}}$ operations.

Lemma 6.1. *Let K be a semiring and $k, k' \in \mathbb{TEC}_K$. Then,*

$$\mathcal{C}_K(k +_{K_\mathcal{P}} k') = \mathcal{C}_K(\mathcal{C}_K(k) +_{K_\mathcal{P}} k')$$
$$\mathcal{C}_K(k \cdot_{K_\mathcal{P}} k') = \mathcal{C}_K(\mathcal{C}_K(k) \cdot_{K_\mathcal{P}} k')$$

Using this lemma, we now prove that for any semiring K, the structure $K_\mathcal{T}$ is also a semiring.

Theorem 6.2. *For any semiring K, structure $K_\mathcal{T}$ is a semiring.*

6.2 Timeslice Operator

We define a timeslice operator for $K_\mathcal{T}$-relations based on the timeslice operator for temporal K-elements. We annotate each tuple in the output of this operator with the result of τ_T applied to the temporal K-element the tuple is annotated with.

Definition 6.2 (Timeslice for $K_\mathcal{T}$-relations). *Let R be a $K_\mathcal{T}$-relation and $T \in \mathbb{T}$. The timeslice operator $\tau_T(R)$ is defined as:*

$$\tau_T(R)(t) = \tau_T(R(t))$$

We now prove that the timeslice operator for temporal K-elements is a homomorphism $K_\mathcal{T} \to K$. Since semiring homomorphisms commute with queries [21], this demonstrates that $K_\mathcal{T}$ equipped with this timeslice operator does fulfill the snapshot-reducibility condition of representations systems (condition 2 of Definition 4.6).

Theorem 6.3. *For any $T \in \mathbb{T}$, the timeslice operator τ_T is a semiring homomorphism from $K_\mathcal{T}$ to K.*

As an example of the application of this homomorphism, consider the period $\mathbb{N}$-relation `works` from our running example as shown on the left of Figure 2. Applying τ_{08} to this relation yields the snapshot shown on the bottom of this figure (three employees work between 8am and 9am out of whom two are specialized). If we evaluate query Q_{onduty} over this snapshot we get the snapshot shown on the right of this figure (the count is 2). By Theorem 6.3 we get the same result if we evaluate Q_{onduty} over the input period $\mathbb{N}$-relation and then apply τ_{08} to the result.

6.3 Encoding of Sequenced K-relations

We now define a bijective mapping $\textsc{Enc}_K$ from sequenced K-relations to $K_\mathcal{T}$-relations. We then prove that the set of $K_\mathcal{T}$-relations together with the timeslice operator for such relations and the mapping $\textsc{Enc}_K{}^{-1}$ (the inverse of $\textsc{Enc}_K$) form a representation system for sequenced K-relations. Intuitively, $\textsc{Enc}_K(R)$ is constructed by assigning each tuple t a temporal K-element where the annotation of the tuple at time T (i.e., $R(T)(t)$) is assigned to a singleton interval $[T, T+1)$. This temporal K-element $\mathcal{T}_{R,t}$ is then coalesced to create a $\mathbb{TEC}_K$ element.

Definition 6.3. *Let K be a semiring and R a sequenced K-relation,* $\textsc{Enc}_K$ *is a mapping from sequenced K-relations to $K_{\mathcal{T}}$-relations defined as follows.*

$$\forall t : \textsc{Enc}_K(R)(t) = \mathcal{C}_K(\mathcal{T}_{R,t})$$

$$\forall t, I : \mathcal{T}_{R,t}(I) = \begin{cases} R(T)(t) & \text{if } I = [T, T+1) \\ 0_K & \text{otherwise} \end{cases}$$

We first prove that this mapping is bijective, i.e., it is invertible, which guarantees that $\textsc{Enc}_K{}^{-1}$ is well-defined and also implies uniqueness (condition 1 of Definition 4.6).

Lemma 6.4. *For any semiring K,* $\textsc{Enc}_K$ *is bijective.*

Next, we have to show that $\textsc{Enc}_K$ preserves the information encoded in a sequenced K-relation R (condition 3 of Definition 4.6). That is, the instance at a time point T represented by R can be extracted from $\textsc{Enc}_K(R)$ using the timeslice operator.

Lemma 6.5. *For any semiring K, sequenced K-relation R, and time point $T \in \mathbb{T}$, we have $\tau_T(\textsc{Enc}_K(R)) = \tau_T(R)$.*

Based on these properties of $\textsc{Enc}_K$ and the fact that the timeslice operator over $K_{\mathcal{T}}$-relations is a homomorphism $K_{\mathcal{T}} \to K$, our main technical result follows immediately. That is, the set of $K_{\mathcal{T}}$-relations equipped with the timeslice operator and $\textsc{Enc}_K{}^{-1}$ is a representation system for positive relational algebra queries ($\mathcal{RA}^+$) over sequenced K-relations.

Theorem 6.6 (Representation System)**.** *Given a semiring K, let $\mathcal{DB}_{K_{\mathcal{T}}}$ be the set of all $K_{\mathcal{T}}$-relations. The triple $(\mathcal{DB}_{K_{\mathcal{T}}}, \textsc{Enc}_K{}^{-1}, \tau)$ is a representation system for $\mathcal{RA}^+$ queries over sequenced K-relations.*

7. COMPLEX QUERIES

Having proven that $K_{\mathcal{T}}$-relations form a representation system for $\mathcal{RA}^+$, we now study extensions for difference and aggregation.

7.1 Difference

Extensions of K-relations for difference have been studied in [18, 2]. Geerts et al. [18] applies an extension of semirings with a monus operation that is defined based on the *natural order* of a semiring and demonstrated how to define a difference operation for K-relations based on the monus operation for semirings where this operations is well-defined. Following the terminology introduced in this work, we refer to semirings with a monus operation as m-semirings. We now prove that if a semiring K has a well-defined monus, then so does $K_{\mathcal{T}}$. From this follows, that for any such K, the difference operation is well-defined for $K_{\mathcal{T}}$. We proceed to show that the timeslice operator is an m-semiring homomorphism, which implies that $K_{\mathcal{T}}$-relations for any m-semiring K form a representation system for $\mathcal{RA}$ (full relational algebra). The definition of a monus operator is based on the so-called natural order $\preceq_K$. For two elements k and k' of a semiring K, $k \preceq_K k' \Leftrightarrow \exists k'' : k +_K k'' = k'$. If $\preceq_K$ is a partial order then K is called *naturally ordered.* For instance, $\mathbb{N}$ is naturally ordered ($\preceq_{\mathbb{N}}$ corresponds to the order of natural numbers) while $\mathbb{Z}$ is not (for any $k, k' \in \mathbb{Z}$ we have $k \preceq_{\mathbb{Z}} k'$). For the monus to be well-defined on K, K has to be naturally ordered and for any $k, k' \in K$, the set $\{k'' \mid k \preceq_K k' +_K k''\}$ has to have a smallest member. For any semiring fulfilling these two conditions, the monus operation $-_K$ is defined as $k -_K k' = k''$ where k'' is the smallest element such that $k \preceq_K k' + k''$. For instance, the monus for $\mathbb{N}$ is the truncating minus: $k -_{\mathbb{N}} k' = max(0, k - k')$.

Theorem 7.1. *For any m-semiring K, semiring $K_{\mathcal{T}}$ has a well-defined monus, i.e., is an m-semiring.*

Let $k -_{K_{\mathcal{P}}} k'$ denote an operation that returns a temporal K-element which assigns to each singleton interval $[T, T+1)$ the result of the monus for K: $\tau_T(k) -_K \tau_T(k')$ (this is k_{pmin} as defined in the proof of Theorem 7.1, see Appendix A). In the proof of Theorem 7.1, we demonstrate that $k -_{K_{\mathcal{T}}} k' = \mathcal{C}_K(k -_{K_{\mathcal{P}}} k')$. Obviously, computing $k -_{K_{\mathcal{P}}} k'$ using singleton intervals is not effective. In our implementation, we use a more efficient way to compute the monus for $K_{\mathcal{T}}$ that is based on normalizing the input temporal K-elements k and k' such that annotations are attached to larger time intervals where $k -_{K_{\mathcal{P}}} k'$ is guaranteed to be constant. Importantly, τ_T is a homomorphism for monus-semiring $K_{\mathcal{T}}$.

Theorem 7.2. *Mapping τ_T is an m-semiring homomorphism.*

For example, consider $Q_{skillreq}$ from Example 1.2 which can be expressed in relational algebra as $\Pi_{skill}(assign) - \Pi_{skill}(worker)$. The $\mathbb{N}_{\mathcal{T}}$-relation corresponding to the period relation `assign` shown in this example annotates each tuple with a singleton temporal $\mathbb{N}$-element mapping the period of this tuple to 1, e.g., `(M1, SP)` is annotated with $\{[03, 12) \mapsto 1\}$. The annotation of result tuple `(SP)` is computed as

$$\begin{aligned}
&(\{[03,12) \mapsto 1\} +_{\mathbb{N}_{\mathcal{T}}} \{[06,14) \mapsto 1\}) \\
&\quad -_{\mathbb{N}_{\mathcal{T}}} (\{[03,10) \mapsto 1\} +_{\mathbb{N}_{\mathcal{T}}} \{[08,16) \mapsto 1\} +_{\mathbb{N}_{\mathcal{T}}} \{[18,20) \mapsto 1\}) \\
=&\{[03,06) \mapsto 1, [06,12) \mapsto 2, [12,14) \mapsto 1\} \\
&\quad -_{\mathbb{N}_{\mathcal{T}}} \{[03,08) \mapsto 1, [08,10) \mapsto 2, [10,16) \mapsto 1, [18,20) \mapsto 1\} \\
=&\{[06,08) \mapsto 1, [10,12) \mapsto 1\}
\end{aligned}$$

As expected, the result is the same as the one from Example 1.2.

7.2 Aggregation

The K-relational framework has previously been extended to support aggregation [3]. This required the introduction of attribute domains which are symbolic expressions that pair values with semiring elements to represent aggregated values. Since the construction used in this work to derive the mathematical structures representing these symbolic expressions is applicable to all semirings, it is also applicable to our period semirings. It was shown that semiring homomorphisms can be lifted to these more complex annotation structures and attribute domains. Thus, the timeslice operator, being a semiring homomorphism, commutes with queries including aggregation, and it follows that using the approach from [3], we can define a representation system for sequenced K-relations under $\mathcal{RA}$ with aggregation, i.e., $\mathcal{RA}^{agg}$.

One drawback of this definition of aggregation over K-relations with respect to our use case is that there are multiple ways of encoding the same sequenced K-relation in this model. That is, we would loose uniqueness of our representation system. Recall that one of our major goals is to implement sequenced query semantics on-top of DBMS using a period multiset encoding of $\mathbb{N}_{\mathcal{T}}$-relations. The symbolic expressions representing aggregation function results are a compact representation which, in case of our interval-temporal semirings, encode how the aggregation function results change over time. However, it is not clear how to effectively encode the symbolic attribute values and comparisons of symbolic expression as multiset semantics relations, and how to efficiently implement our sequenced semantics over this encoding. Nonetheless, for $\mathbb{N}$, we can apply a simpler definition of aggregation that returns a $K_{\mathcal{T}}$ relation and is also a representation system. For simplicity, we define aggregation ${}_G\gamma_{f(A)}(R)$ grouping on G to compute a single aggregation function f over the values of an attribute A. For convenience, aggregation without group-by, i.e., $\gamma_{f(A)}(R)$ is expressed using an empty group-by list.

Definition 7.1 (Aggregation)**.** *Let R be a $\mathbb{N}_{\mathcal{T}}$ relation. Operator ${}_G\gamma_{f(A)}(R)$ groups the input on a (possibly empty) list of attributes $G = (g_1, \ldots, g_n)$ and computes aggregation function f over the values of attribute A. This operator is defined as follows:*

$$
{}_G\gamma_{f(A)}(R)(t) = \mathcal{C}_{\mathbb{N}}(k_{R,t})
$$

$$
k_{R,t}(I) = \begin{cases} 1 & \textit{if } \exists T : I = [T, T{+}1) \wedge t \in {}_G\gamma_{f(A)}(\tau_T(R)) \\ 0 & \textit{otherwise} \end{cases}
$$

In the output of the aggregation operator, each tuple t is annotated with a $\mathbb{N}$-coalesced temporal $\mathbb{N}$-element which is constructed from singleton intervals. A singleton interval $I = [T, T + 1)$ is mapped to 1 if evaluating the aggregation over the multiset relation corresponding to the snapshot at T returns tuple t. We now demonstrate that $\mathbb{N}_{\mathcal{T}}$ using this definition of aggregation is a representation system for snapshot $\mathbb{N}$-relations.

Theorem 7.3. *$\mathbb{N}_{\mathcal{T}}$-relations form a representation system for sequenced $\mathbb{N}$-relations and $\mathcal{RA}^{agg}$ queries using aggregation according to Definition 7.1.*

8. SQL PERIOD RELATION ENCODING

While provably correct, the annotation structure that we have defined is quite complex in nature raising concerns on how to efficiently implement it. In this section we demonstrate that $\mathbb{N}_{\mathcal{T}}$-relations (multisets) can be encoded as SQL period relations (as shown on the top of Figure 2). Recall that SQL period relations are multiset relations where the validity time interval (period) of a tuple is stored in an interval-valued attribute (or as two attributes which store the interval's end points). Queries over $\mathbb{N}_{\mathcal{T}}$ are then translated into non-temporal multiset queries over this encoding. In addition to employing a proven and simple representation of time this enables our approach to run sequenced queries over such relations without requiring any preprocessing and to implement our ideas on top of a standard DBMS. For convenience we represent SQL period relations using non-temporal $\mathbb{N}$-relations in the definitions. SQL period relations can be obtained based on the well-known correspondence between multiset relations and $\mathbb{N}$-relations: we duplicate each tuple based on the multiplicity recorded in its annotation. To encode $\mathbb{N}_{\mathcal{T}}$-relations as $\mathbb{N}$-relations we introduce an invertible mapping PERIODENC. We rewrite queries with $\mathbb{N}_{\mathcal{T}}$-semantics into non-temporal queries with $\mathbb{N}$-semantics over this encoding using a rewriting function REWR. This is illustrated in the commutative diagram below.

$$
\begin{tikzcd}
\boxed{R} \arrow[r, "\textsc{PeriodEnc}"] \arrow[d, "Q"'] & \boxed{R'} \arrow[d, "Q' = \textsc{Rewr}(Q)"] \\
\boxed{Q(R)} & \boxed{Q'(R')} \arrow[l, "\textsc{PeriodEnc}^{-1}"]
\end{tikzcd}
\qquad (1)
$$

Our encoding represents a tuple t annotated with a temporal element $\mathcal{T}$ as a set of tuples, one for each interval I which is assigned a non-zero value by $\mathcal{T}$. For each such interval, the interval's end points are stored in two attributes A_{begin} and A_{end}, which are appended to the schema of t. Again, we use $t \mapsto k$ to denote that tuple t is annotated with k and $\mathcal{U}$ to denote a universal domain of values. We use $SCH(R)$ to denote the schema of relation R and $arity(R)$ to denote its arity (the number of attributes in the schema).

Definition 8.1 (Encoding as SQL Period Relations)**.** *PERIODENC is a function from $\mathbb{N}_{\mathcal{T}}$-relations to $\mathbb{N}$-relations. Let R be a $\mathbb{N}_{\mathcal{T}}$ relation with schema $SCH(R) = \{A_1, \ldots, A_n\}$. The schema of* PERIODENC(R) *is $\{A_1, \ldots, A_n, A_{begin}, A_{end}\}$. Let R' be* PERIODENC(R) *for some $\mathbb{N}_{\mathcal{T}}$-relation.* PERIODENC *and its inverse are defined as follows:*

$$
\textsc{PeriodEnc}(R) = \bigcup_{t \in \mathcal{U}^{arity(R)}} \bigcup_{I \in \mathbb{I}} \{(t, I^+, I^-) \mapsto R(t)(I)\}
$$

$$
\textsc{PeriodEnc}^{-1}(R') = \bigcup_{t \in \mathcal{U}^{arity(R)}} \{t \mapsto \mathcal{T}_{R',t}\}
$$

$$
\forall I \in \mathbb{I} : \mathcal{T}_{R',t}(I) = R'(t_I) \textit{ for } t_I = (t, I^+, I^-)
$$

Before we define the rewriting REWR that reduces a query Q with $\mathbb{N}_{\mathcal{T}}$ semantics to a query with $\mathbb{N}$ semantics, we introduce two operators that we will make use of in the reduction. The $\mathbb{N}$-coalesce operator applies $\mathcal{C}_{\mathbb{N}}$ to the annotation of each tuple in its input.

Definition 8.2 (Coalesce Operator)**.** *Let R be* PERIODENC(R') *for some $\mathbb{N}_{\mathcal{T}}$-relation R'. The coalesce operator $\mathcal{C}(R)$ is defined as:*

$$
\mathcal{C}(R) = \textsc{PeriodEnc}(R')
$$

$$
\forall t : R'(t) = \mathcal{C}_{\mathbb{N}}(\textsc{PeriodEnc}^{-1}(R)(t))
$$

The split operator $\mathcal{N}_G(R, S)$ splits the intervals in the temporal elements annotating a tuple t based on the union of all interval end points from annotations of tuples t' which agree with t on attributes G. Inputs R and S have to be union compatible. The effect of this operator is that all pairs of intervals mapped to non-zero elements are either the same or are disjoint. This operator has been applied in [15, 17] and in [11, 44]. We use it to implement snapshot-reducible aggregation and difference over intervals instead of single snapshots as in Section 7. Recall that in Section 7, the monus (difference) and aggregation were defined in a point-wise manner. The split operator allows us to evaluate these operations over intervals directly by generating tuples with intervals for which the result of these operations is guaranteed to be constant.

Definition 8.3 (Split Operator)**.** *The split operator $\mathcal{N}_G(R_1, R_2)$ takes as input two $\mathbb{N}$-relations R_1 and R_2 that are encodings of $\mathbb{N}_{\mathcal{T}}$-relations. For a tuple t in such an encoding let $I(t) = [t.A_{begin}, t.A_{end})$. The split operator is defined as:*

$$
\mathcal{N}_G(R_1, R_2)(t) = \textit{split}(t, R_1, EP_G(R_1 \cup R_2, t))
$$

$$
EP_G(R, t) = \bigcup_{t' \in R : t'.G = t.G \wedge R(t') > 0} \{t'.A_{begin}\} \cup \{t'.A_{end}\}
$$

$$
\textit{split}(t, R, EP) = \sum_{t' : I(t) \subseteq I(t') \wedge I(t) \in EPI(t, EP)} R(t')
$$

$$
\begin{aligned}
EPI(t, EP) = \{[T_b, T_e) \mid\ & T_b <_{\mathbb{T}} T_e \wedge T_b \in EP \wedge \\
& (T_e \in EP \vee T_e = T_{max}) \wedge \\
& \nexists T' \in EP : T_b <_{\mathbb{T}} T' <_{\mathbb{T}} T_e\}
\end{aligned}
$$

Note that the PERIODENC and PERIODENC^{-1} mappings are only used in the definitions of the coalesce and split algebra operators for ease of presentation. These operators can be implemented as SQL queries executed over an PERIODENC-encoded relation.

Definition 8.4 (Query Rewriting)**.** *We use $overlaps(Q_1, Q_2)$ as a notational shortcut for $Q_1.A_{begin} < Q_2.A_{end} \wedge Q_2.A_{begin} < Q_1.A_{end}$. The definition of the rewriting* REWR *is shown in Figure 4. Let 0_f denote the result of an aggregation function over an empty input ($0_{count} = 0$ and $0_a = null$ for any other aggregation function a) and $\{t\}$ denote a constant relation with a single tuple t annotated with* 1.

$$\underline{\textsc{Rewr}(R)} = R \qquad \underline{\textsc{Rewr}(\sigma_\theta(Q))} = \mathcal{C}(\sigma_\theta(\textsc{Rewr}(Q))) \qquad \underline{\textsc{Rewr}(\Pi_A(Q))} = \mathcal{C}(\Pi_{A,A_{begin},A_{end}}(\textsc{Rewr}(Q)))$$

$$\underline{\textsc{Rewr}(Q_1 \bowtie_\theta Q_2)} = \mathcal{C}(\Pi_{SCH(Q_1 \bowtie_\theta Q_2),max(Q_1.A_{begin},Q_2.A_{begin}),min(Q_1.A_{end},Q_2.A_{end})}(\textsc{Rewr}(Q_1) \bowtie_{\theta \wedge overlaps(Q_1,Q_2)} \textsc{Rewr}(Q_2)))$$

$$\underline{\textsc{Rewr}(Q_1 - Q_2)} = \mathcal{C}(\mathcal{N}_{SCH(Q_1)}(\textsc{Rewr}(Q_1), \textsc{Rewr}(Q_2)) - \mathcal{N}_{SCH(Q_2)}(\textsc{Rewr}(Q_2), \textsc{Rewr}(Q_1)))$$

$$\underline{\textsc{Rewr}(\gamma_{f(A)}(Q))} = \mathcal{C}({}_{A_{begin},A_{end}}\gamma_{f(A)}(\mathcal{N}_\emptyset(\textsc{Rewr}(Q) \cup \{(0_f, T_{min}, T_{max})\}, \textsc{Rewr}(Q))))$$

$$\underline{\textsc{Rewr}({}_G\gamma_{f(A)}(Q))} = \mathcal{C}({}_{G,A_{begin},A_{end}}\gamma_{f(A)}(\mathcal{N}_G(\textsc{Rewr}(Q), \textsc{Rewr}(Q)))) \qquad \underline{\textsc{Rewr}(Q_1 \cup Q_2)} = \mathcal{C}(\textsc{Rewr}(Q_1) \cup \textsc{Rewr}(Q_2))$$

Figure 4: Rewriting REWR that reduces queries over $\mathbb{N}_\mathcal{T}$ to queries over a multiset encoding produced by PERIODENC.

Theorem 8.1. *The commutative diagram in Equation* (1) *holds.*

9. IMPLEMENTATION

We have implemented the encoding and rewriting introduced in the previous section in a middleware which supports sequenced multiset semantics through an extension of SQL. To instruct the system to interpret a subquery using sequenced semantics, the user encloses the subquery in a `SEQ VT (...)` block. We assume that the inputs to a sequenced query are encoded as period multiset relations, i.e., each relation has two temporal attributes that store the begin and end timestamp of their validity interval. For each relation access within a `SEQ VT` block, the user has to specify which attributes store the period of a tuple.

Our coalescing and split operators can be expressed in SQL. Thus, a straightforward way of incorporating these operators into the compilation process is to devise additional rewrites that produce the relational algebra code for these operators where necessary. However, preliminary experiments demonstrated that a naive implementation of these operators is prohibitively expensive.

We address this problem in two ways. First, we observe that it is sufficient to apply coalesce as a last step in a query instead of applying it as part of every operator rewrite. Applying this optimization, the rewritten version of a query will only contain one coalesce operator. Recall from Lemma 6.1 that coalescing can be redundantly pushed into the addition and multiplication operations of interval-temporal semirings, e.g., $\mathcal{C}_K(k +_{K_\mathcal{P}} k') = \mathcal{C}_K(\mathcal{C}_K(k) +_{K_\mathcal{P}} k')$. We prove that this Lemma also holds for monus in Appendix C. Interpreting this equivalence from right to left and applying it repeatedly to a semiring expression e, e can be rewritten into an equivalent expression of the form $\mathcal{C}_K(e')$, where e' is an expression that only uses operations $+_{K_\mathcal{P}}$, $\cdot_{K_\mathcal{P}}$, $-_{K_\mathcal{P}}$. Since relational algebra over K-relations is defined by applying multiplication, addition, and monus to input annotations, this implies that it is sufficient to apply coalescing only as a final operation in a query. For an example and additional discussion see Appendix C

Furthermore, we developed an optimized implementation of coalesce using SQL analytical functions to count the number of open intervals per time point, determine change points based on differences between these counts, and then only output maximal intervals using a filter step. For splitting aggregation inputs, it turned out to be most effective to pre-aggregate the input and then compute the final aggregation results during the split step. We apply a similar optimization for difference.

10. EXPERIMENTS

In our experimental evaluation we focus on two aspects. First, we evaluate the cost of our SQL implementation of $\mathbb{N}$-coalescing (multiset coalescing). Then, we evaluate the performance of sequenced queries with our approach over three DBMSs and compare it against native implementations of sequenced semantics that are available in two of these systems (using our implementation of coalescing to produce a coalesced result).

10.1 Workloads and Experimental Setup

Dataset. We use the MySQL Employees dataset (`https://github.com/datacharmer/test_db`), which contains ≈4 million records. The schema consists of the following six period tables: table `employee` stores basic information about employees; table `departments` stores department information; table `titles` stores the job titles for employees; table `salaries` stores employee salaries; table `dept_manager` stores which employee manages which department; and table `dept_emp` stores which employee is working for which department.

Workload. We have created a workload consisting of 10 queries to evaluate the efficiency of sequenced queries. Queries `join-1` to `join-4` are join queries, `agg-1` to `agg-3` are aggregation-heavy queries, `agg-join` is a join with an aggregation value, and `diff-1` and `diff-2` use difference. Furthermore, we use one query template varying the selectivity to evaluate the performance of coalescing. `C-Sn` denotes the variant of this query that returns approximately nK rows, e.g., `C-S1` returns 1,000 rows. More detailed descriptions of these queries are provided in Appendix B. The number of rows returned by these queries over the dataset are shown in Table 2.

Table 2: Size of query results.

join-1	join-2	join-3	join-4	agg-1	agg-2	agg-3	agg-join	diff-1	diff-2
2.8M	28.3M	10	177	57.4k	177	210	260	300k	2.8M

Systems. We ran experiments on three different database management systems: a version of Postgres (*PG*) with native support for temporal operators as described in [15, 17]; a commercial DBMS, *DBX*, with native support for sequenced semantics (only available as a virtual machine); and a commercial DBMS, *DBY*, without native support for sequenced semantics. We used our approach to translate sequenced queries into standard SQL queries and ran the translated queries on all three systems (denoted as *PG-Seq*, *DBX-Seq*, and *DBY-Seq*). For PG and DBX, we ran the queries also with the native solution for sequenced semantics paired with our implementation of coalescing to produce a coalesced result (referred to as *PG-Nat* and *DBX-Nat*). As explained in Section 2, no system correctly implements sequenced multiset semantics for difference and aggregation, and many systems do not support sequenced semantics for these operators at all. DBX and the variant of Postgres we are using both support sequenced aggregation, however, their implementations are not snapshot-reducible. DBX does not support sequenced difference, whereas our Postgres version implements temporal difference with set semantics. Despite such differences, the experimental comparison allows us to understand the performance impact of our provably correct approach.

All experiments were executed on a machine with 2 AMD Opteron 4238 CPUs, 128GB RAM, and a hardware RAID with 4 $\times$ 1TB 72.K HDs in RAID 5.

10.2 Performance Results

10.2.1 Multiset Coalescing

To evaluate the performance of coalescing, we use a selection query that returns employees that earn more than a specific salary and materialize the result as a table. The selectivity varies from 1K to 3M rows. We then evaluate the query **SELECT** * **FROM** ... over the materialized tables under sequenced semantics in order to measure the cost of coalescing in isolation. Figure 5 shows the results of this experiment. The runtime of coalescing is linear in the input size for all three systems. Even though the theoretical worst-case complexity of the sorting step, which is applied by all systems to evaluate the analytics functions that we exploit in our SQL-based implementation of multiset coalescing, is $\mathcal{O}(n \cdot log(n))$, an inspection of the execution plans revealed that the sorting step only amounts to 5%-10% of the execution time (for all selectivities) and, hence, is not a dominating factor.

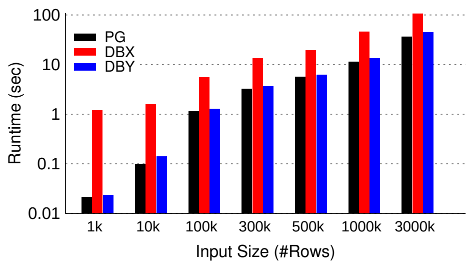


Figure 5: Multiset coalescing for varying input size.

10.2.2 Sequenced Semantics Queries

Table 3 provides an overview about the the performance results for our sequenced query workloads.

Table 3: Runtimes (sec) of sequenced queries: **N/A** = not supported, **OOTS** = system ran out of temporary space (2GB).

Query	**PG-Seq**	**PG-Nat**	**DBX-Seq**	**DBX-Nat**	**DBY-Seq**
`join-1`	91.97	118.01	118.95	116.03	64.00
`join-2`	1543.81	888.13	1569.45	1200.36	763.70
`join-3`	0.01	4.91	0.55	0.43	0.01
`join-4`	0.52	12.85	0.83	0.60	0.22
`agg-1`	7.02	5980.85	56.47	**OOTS**	5.24
`agg-2`	0.06	10.31	0.82	0.82	0.01
`agg-3`	1.42	0.02	0.78	0.55	0.01
`agg-join`	6643.61	19195.03	**OOTS**	**OOTS**	7555.97
`diff-1`	14.18	6.88	30.15	**N/A**	10.29
`diff-2`	63.58	79.63	129.87	**N/A**	61.90

Join Queries. The performance of our approach for join queries is comparable with the native implementation in *PG-Nat*. For join queries with larger intermediate results (`join-2`), the native implementation outperforms our approach by ≈73%. Running the queries produced by our approach in *DBY* is slightly faster than both. *DBX-Nat* uses merge joins for temporal joins, while both *PG* and *DBY* use a hash-join on the non-temporal part of the join condition. The result is that *DBX-Nat* significantly outperforms the other methods for temporal join operations. However, the larger cost for the SQL-based coalescing implementation in this system often outweighs this effect. This demonstrates the potential for improving our approach by making use of native implementations of temporal operators in our rewrites for operators that are compatible with our semantics (note that joins are compatible).

Aggregation Queries. Our approach outperforms the native implementations of sequenced semantics on all systems by several orders of magnitude for aggregation queries as long as the aggregation input exceeds a certain size (`agg-1` and `agg-2`). Our approach as well as the native approaches split the aggregation input which requires sorting and then apply a standard aggregation operator to compute the temporal aggregation result. The main reason for the large performance difference is that the SQL code we generate for a sequenced aggregation includes several levels of pre-aggregation that are intertwined with the split operator. Thus, for our approach the sorting step for split is applied to a typically much smaller pre-aggregated dataset. This turned out to be quite effective. The only exception is if the aggregation input is very small (`agg-3`) in which case an efficient implementation of split (as in *PG-Nat*) outweighs the benefits of pre-aggregation. Query `agg-1` did not finish on *DBX-Nat* as it exceeded the DBMS's temporary space (memory allocated for intermediate results) of 2GB.

Mixed Aggregation and Join. Query `agg-join` applies an aggregation over the result of several joins. Our approach is more effective, in particular for the aggregation part of this query, compared to *PG-Nat*. This query did not finish on *DBX* due to the 2GB temporary space restriction per query imposed by the DBMS.

Difference Queries. For difference queries we could only compare our approach against *PG-Nat*, since *DBX-Nat* does not support difference in sequenced queries. Note that, *PG-Nat* applies set difference while our approach supports multiset difference. While our approach is less effective for `diff-1` which contains a single difference operator, we outperform *PG-Nat* on `diff-2`.

10.3 Summary

Our experiments demonstrate that an SQL-based implementation of multiset coalescing is feasible – exhibiting runtimes linear in the size of the input, albeit with a relatively large constant factor. We expect that it would be possible to significantly reduce this factor by introducing a native implementation of this operator. Using pre-aggregation during splitting, our approach significantly outperforms native implementations for aggregation queries. DBX uses merge joins for temporal joins (interval overlap joins) which is significantly more efficient than hash joins which are employed by Postgres and DBY. This shows the potential of integrating such specialized operators with our approach in the future. For example, we could compile sequenced queries into SQL queries that selectively employ the temporal extensions of a system like DBX.

11. CONCLUSIONS AND FUTURE WORK

We present the first provably correct interval-based representation system for sequenced semantics over multiset relations and its implementation in a database middleware. We achieve this goal by addressing a more general problem: snapshot-reducibility for temporal K-relations. Our solution is a uniform framework for evaluation of queries under sequenced semantics over an interval-based encoding of temporal K-relations for any semiring K. That is, in addition to sets and multisets, the framework supports sequenced temporal extensions of probabilistic databases, databases annotated with provenance, and many more. In future work, we will study how to extend our approach for updates over annotated relations, will study its applicability for combining probabilistic and temporal query processing, and investigate implementations of split and K-coalescing inside a database kernel.

12. REFERENCES

[1] Mohammed Al-Kateb, Ahmad Ghazal, Alain Crolotte, Ramesh Bhashyam, Jaiprakash Chimanchode, and Sai Pavan Pakala. Temporal query processing in Teradata. In *EDBT*, pages 573–578, 2013.

[2] Yael Amsterdamer, Daniel Deutch, and Val Tannen. On the limitations of provenance for queries with difference. In *TaPP*, 2011.

[3] Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *PODS*, pages 153–164, 2011.

[4] Michael H. Böhlen, Johann Gamper, and Christian S. Jensen. Multi-dimensional aggregation for temporal data. In *EDBT*, pages 257–275, 2006.

[5] Michael H. Böhlen, Johann Gamper, Christian S. Jensen, and Richard T. Snodgrass. Sql-based temporal query languages. In *Encyclopedia of Database Systems*, pages 2762–2768. 2009.

[6] Michael H. Böhlen and Christian S. Jensen. Sequenced semantics. In *Encyclopedia of Database Systems*, pages 2619–2621. 2009.

[7] Michael H. Böhlen, Christian S. Jensen, and Richard T. Snodgrass. Evaluating and enhancing the completeness of tsql2. Technical Report TR 95-5, Computer Science Department, University of Arizona, 1995.

[8] Michael H. Böhlen, Christian S. Jensen, and Richard T. Snodgrass. Temporal statement modifiers. *ACM Trans. Database Syst.*, 25(4):407–456, 2000.

[9] Michael H. Böhlen, Richard T. Snodgrass, and Michael D. Soo. Coalescing in temporal databases. In *VLDB*, pages 180–191, 1996.

[10] Panagiotis Bouros and Nikos Mamoulis. A forward scan based plane sweep algorithm for parallel interval joins. *PVLDB*, 10(11):1346–1357, 2017.

[11] Ivan T. Bowman and David Toman. Optimizing temporal queries: efficient handling of duplicates. *Data Knowl. Eng.*, 44(2):143–164, 2003.

[12] Francesco Cafagna and Michael H. Böhlen. Disjoint interval partitioning. *VLDB J.*, 26(3):447–466, 2017.

[13] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[14] Anish Das Sarma, Martin Theobald, and Jennifer Widom. Live: A lineage-supported versioned dbms. In *SSDBM*, pages 416–433, 2010.

[15] Anton Dignös, Michael H. Böhlen, and Johann Gamper. Temporal alignment. In *SIGMOD*, pages 433–444, 2012.

[16] Anton Dignös, Michael H. Böhlen, and Johann Gamper. Overlap interval partition join. In *SIGMOD*, pages 1459–1470, 2014.

[17] Anton Dignös, Michael H. Böhlen, Johann Gamper, and Christian S. Jensen. Extending the kernel of a relational DBMS with comprehensive support for sequenced temporal queries. *ACM Trans. Database Syst.*, 41(4):26:1–26:46, 2016.

[18] F. Geerts and A. Poggi. On database query languages for K-relations. *Journal of Applied Logic*, 8(2):173–185, 2010.

[19] Boris Glavic, Renée J Miller, and Gustavo Alonso. Using SQL for efficient generation and querying of provenance information. In *In search of elegance in the theory and practice of computation: a Festschrift in honour of Peter Buneman*, pages 291–320. 2013.

[20] Todd J. Green. Containment of Conjunctive Queries on Annotated Relations. In *ICDT*, pages 296–309, 2009.

[21] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[22] Christian S. Jensen and Richard T. Snodgrass. Snapshot equivalence. In *Encyclopedia of Database Systems*, page 2659. 2009.

[23] Christian S. Jensen and Richard T. Snodgrass. Temporal query languages. In *Encyclopedia of Database Systems*, pages 3009–3012. 2009.

[24] Christian S. Jensen and Richard T. Snodgrass. Timeslice operator. In *Encyclopedia of Database Systems*, pages 3120–3121. 2009.

[25] Martin Kaufmann, Amin Amiri Manjili, Panagiotis Vagenas, Peter M. Fischer, Donald Kossmann, Franz Färber, and Norman May. Timeline index: a unified data structure for processing queries on temporal data in SAP HANA. In *SIGMOD*, pages 1173–1184, 2013.

[26] Egor V. Kostylev and Peter Buneman. Combining dependent annotations for relational algebra. In *ICDT*, pages 196–207, 2012.

[27] Krishna G. Kulkarni and Jan-Eike Michels. Temporal features in SQL: 2011. *SIGMOD Record*, 41(3):34–43, 2012.

[28] Nikos A. Lorentzos and Yannis G. Mitsopoulos. SQL extension for interval data. *IEEE Trans. Knowl. Data Eng.*, 9(3):480–499, 1997.

[29] Microsoft. Sql server 2016 - temporal tables. `https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables`, 2016.

[30] Oracle. Database development guide - temporal validity support. `https://docs.oracle.com/database/121/ADFNS/adfns_design.htm#ADFNS967`, 2016.

[31] Danila Piatov and Sven Helmer. Sweeping-based temporal aggregation. In *SSTD*, pages 125–144, 2017.

[32] Danila Piatov, Sven Helmer, and Anton Dignös. An interval join optimized for modern hardware. In *ICDE*, pages 1098–1109, 2016.

[33] Markus Pilman, Martin Kaufmann, Florian Köhl, Donald Kossmann, and Damien Profeta. Partime: Parallel temporal aggregation. In *SIGMOD*, pages 999–1010, 2016.

[34] PostgreSQL. Documentation manual postgresql - range types. `https://www.postgresql.org/docs/current/static/rangetypes.html`, 2012.

[35] Cynthia Saracco, Matthias Nicola, and Lenisha Gandhi. A matter of time: Temporal data management in db2 10. `http://www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata/dm-1204db2temporaldata-pdf.pdf`, 2012.

[36] Richard T. Snodgrass. The temporal query language tquel. *ACM Trans. Database Syst.*, 12(2):247–298, 1987.

[37] Richard T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. 1995.

[38] Richard T. Snodgrass, Ilsoo Ahn, Gad Ariav, Don S. Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käfer, Nick Kline, Krishna G. Kulkarni, T. Y. Cliff Leung, Nikos A. Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo, and Suryanarayana M. Sripada. TSQL2 language specification.

SIGMOD Record, 23(1):65–86, 1994.

[39] Richard T Snodgrass, Michael H Böhlen, Christian S Jensen, and Andreas Steiner. Adding valid time to sql/temporal. *ANSI X3H2-96-501r2, ISO/IEC JTC*, 1, 1996.

[40] Michael D Soo, Christian S Jensen, and Richard T Snodgrass. An algebra for tsql2. In *The TSQL2 temporal query language*, pages 505–546. 1995.

[41] Andreas Steiner. *A generalisation approach to temporal data models and their implementations*. PhD thesis, ETH Zurich, 1998.

[42] Teradata. Teradata database - temporal table support. `http://www.info.teradata.com/download.cfm?ItemID=1006923`, Jun 2015.

[43] David Toman. Point vs. interval-based query languages for temporal databases. In *PODS*, pages 58–67, 1996.

[44] David Toman. Point-based temporal extensions of sql and their efficient implementation. In *Temporal databases: research and practice*, pages 211–237. 1998.

[45] Xin Zhou, Fusheng Wang, and Carlo Zaniolo. Efficient temporal coalescing query support in relational database systems. In *DEXA*, pages 676–686, 2006.

APPENDIX

A. PROOFS

Proof of Lemma 5.1. Equivalence preservation: Proven by contradiction. Assume that $\exists T : \tau_T(\mathcal{T}) \neq \tau_T(\mathcal{C}_K(\mathcal{T}))$. We have to distinguish two cases. If $T \in CP(\mathcal{T})$, then by definition of K-coalesce we have the contradiction: $\tau_T(\mathcal{C}_K(\mathcal{T})) = \tau_T(\mathcal{T})$. If $T \notin CP(\mathcal{T})$, then let T' be the largest change point that is smaller than T (this point has to exist). From the definition of change points follows that $\tau_T = \tau_{T'}$. By construction there has to exists exactly one interval overlapping T that is assigned a non-zero value in $\mathcal{C}_K(\mathcal{T})$ and this interval starts in T'. Hence, we have the contradiction.
Uniqueness: Note that change points are defined using τ_T only and $(\forall T \in \mathbb{T} : \tau_T(\mathcal{T}_1) = \tau_T(\mathcal{T}_2)) \Leftrightarrow \mathcal{T}_1 \sim \mathcal{T}_2$. Since the result of coalescing is uniquely determined by the change points of a temporal element it follows that $\mathcal{T}_1 \sim \mathcal{T}_2 \Leftrightarrow \mathcal{C}_K(\mathcal{T}_1) = \mathcal{C}_K(\mathcal{T}_2)$
Idempotence: Idempotence follows from the other two properties. If we substitute $\mathcal{T}$ and $\mathcal{C}_K(\mathcal{T})$ for $\mathcal{T}_1$ and $\mathcal{T}_2$ in the uniqueness condition, we get idempotence: $\mathcal{C}_K(\mathcal{C}_K(\mathcal{T})) = \mathcal{C}_K(\mathcal{T})$. □

Proof of Lemma 6.1. Push Through Addition: We prove this part by proving that for any k'' if $k \sim k'$ then $(k +_{K_\mathcal{P}} k'') \sim (k' +_{K_\mathcal{P}} k'')$. We have to show that for all $T \in \mathbb{T}$ we have $\tau_T(k + k'') = \tau_T(k' + k'')$. Substituting definitions we get:

$$\sum_{T \in I}(k(I) +_K k''(I)) = (\sum_{T \in I} k(I)) +_K (\sum_{T \in I} k''(I))$$

Using $k \sim k'$ and substituting the definition of $\sim$, i.e., $\sum_{T \in I} k(I) = \sum_{T \in I} k'(I)$, we get:

$$= (\sum_{T \in I} k'(I)) +_K (\sum_{T \in I} k''(I)) = \sum_{T \in I}(k'(I) +_K k''(I))$$

Push Through Multiplication: Analog to the proof for addition, we prove this part by showing that snapshot equivalence of inputs implies snapshot equivalence of outputs for multiplication.

$$\tau_T(k \cdot_{K_\mathcal{P}} k'') = \sum_{\forall I', I'' : I = I' \cap I'' \wedge T \in I} k(I') \cdot_K k''(I'')$$

Based on the fact that timeslice is a homomorphism $K_\mathcal{T} \to K$ which we will prove in Theorem 6.3, time slice commutes with multiplication and addition:

$$\begin{aligned} &=(\sum_{\forall I \wedge T \in I} k(I)) \cdot_K (\sum_{\forall I \wedge T \in I} k''(I)) \\ &=(\sum_{\forall I \wedge T \in I} k'(I)) \cdot_K (\sum_{\forall I \wedge T \in I} k''(I)) \\ &=\tau_T(k' \cdot_{K_\mathcal{P}} k'') \end{aligned}$$ □

Proof of Theorem 6.2. We have to show that the structure we have defined obeys the laws of commutative semirings. Since the elements of $K_\mathcal{T}$ are functions, it suffices to show $k(I) = k'(I)$ for every $I \in \mathbb{I}$ to prove that $k = k'$. For all $k, k' \in K_\mathcal{T}$ and $I \in \mathbb{I}$:
Addition is commutative:

$$(k +_{K_\mathcal{P}} k')(I) = k(I) +_K k'(I) = k'(I) +_K k(I) = (k +_{K_\mathcal{P}} k')(I)$$
$$k +_{K_\mathcal{T}} k' = \mathcal{C}_K(k +_{K_\mathcal{P}} k') = \mathcal{C}_K(k' +_{K_\mathcal{P}} k) = k' +_{K_\mathcal{T}} k$$

Addition is associative:

$$((k +_{K_\mathcal{P}} k') +_{K_\mathcal{P}} k'')(I) = (k(I) +_K k'(I)) +_K k''(I)$$
$$= k(I) +_K (k'(I) +_K k''(I)) = (k +_{K_\mathcal{P}} (k' +_{K_\mathcal{P}} k''))(I)$$

$$(k +_{K_\mathcal{T}} k') = \mathcal{C}_K(k +_{K_\mathcal{P}} k') = \mathcal{C}_K(k' +_{K_\mathcal{P}} k) = (k' +_{K_\mathcal{T}} k)$$

Zero is neutral element of addition:

$$(k +_{K_\mathcal{P}} 0_{K_\mathcal{T}})(I) = k(I) +_K 0_{K_\mathcal{T}}(I) = k(I) +_K 0_K = k(I)$$
$$k +_{K_\mathcal{T}} 0_{K_\mathcal{T}} = \mathcal{C}_K(k +_{K_\mathcal{P}} 0_{K_\mathcal{T}}) = \mathcal{C}_K(k) = k$$

Multiplication is commutative:

$$\begin{aligned} (k \cdot_{K_\mathcal{P}} k')(I) &= \sum_{\forall I', I'' : I = I' \cap I''} k(I') \cdot_K k'(I'') \\ &= \sum_{\forall I'', I' : I = I'' \cap I'} k(I'') \cdot_K k'(I') \\ &= \sum_{\forall I', I'' : I = I' \cap I''} k'(I') \cdot_K k(I'') = (k' \cdot_{K_\mathcal{P}} k)(I) \end{aligned}$$
$$k \cdot_{K_\mathcal{T}} k' = \mathcal{C}_K(k \cdot_{K_\mathcal{P}} k') = \mathcal{C}_K(k' \cdot_{K_\mathcal{P}} k) = k' \cdot_{K_\mathcal{T}} k$$

Multiplication is associative:

$$\begin{aligned} &((k \cdot_{K_\mathcal{P}} k') \cdot_{K_\mathcal{P}} k'')(I) \\ &= \sum_{\forall I_1, I_2 : I = I_1 \cap I_2} (\sum_{\forall I_3, I_4 : I_1 = I_3 \cap I_4} k(I_3) \cdot_K k'(I_4)) \cdot_K k''(I_2) \\ &= \sum_{\forall I_1, I_2 : I = I_1 \cap I_2} \sum_{\forall I_3, I_4 : I_1 = I_3 \cap I_4} (k(I_3) \cdot_K k'(I_4) \cdot_K k''(I_2)) \\ &= \sum_{\forall I_1, I_2, I_3 : I = I_1 \cap I_2 \cap I_3} k(I_1) \cdot_K k'(I_2) \cdot_K k''(I_3) \\ &= \sum_{\forall I_1, I_2 : I = I_1 \cap I_2} k(I_1) \cdot (\sum_{\forall I_3, I_4 : I_2 = I_3 \cap I_4} k'(I_3) \cdot_K k''(I_4)) \\ &= (k \cdot_{K_\mathcal{P}} (k' \cdot_{K_\mathcal{P}} k''))(I) \end{aligned}$$

$$\begin{aligned} &(k \cdot_{K_\mathcal{T}} k') \cdot_{K_\mathcal{T}} k' \\ &= \mathcal{C}_K(\mathcal{C}_K(k \cdot_{K_\mathcal{P}} k') \cdot_{K_\mathcal{P}} k'') \\ &= \mathcal{C}_K(k \cdot_{K_\mathcal{P}} k' \cdot_{K_\mathcal{P}} k'') \\ &= \mathcal{C}_K(k \cdot_{K_\mathcal{P}} \mathcal{C}_K(k' \cdot_{K_\mathcal{P}} k'')) \\ &= k \cdot_{K_\mathcal{T}} (k' \cdot_{K_\mathcal{T}} k'') \end{aligned}$$

One is neutral element of multiplication:

$$(k \cdot_{K_{\mathcal{P}}} 1_{K_{\mathcal{T}}})(I) = \sum_{\forall I', I'': I = I' \cap I''} k(I') \cdot_K 1_{K_{\mathcal{T}}}(I'')$$
$$= k(I) \cdot_{K_{\mathcal{P}}} 1_{K_{\mathcal{T}}}([t_{min}, t_{max})) = k(I) \cdot_K 1_K$$
$$= k(I)$$
$$k \cdot_{K_{\mathcal{T}}} 1_{K_{\mathcal{T}}}$$
$$= \mathcal{C}_K(k \cdot_{K_{\mathcal{P}}} 1_{K_{\mathcal{T}}}) = \mathcal{C}_K(k) = k$$

Distributivity:

$$(k \cdot_{K_{\mathcal{P}}} (k' +_{K_{\mathcal{P}}} k''))(I)$$
$$= \sum_{\forall I', I'': I = I' \cap I''} k(I') \cdot_K (k'(I'') +_K k''(I''))$$
$$= \sum_{\forall I', I'': I = I' \cap I''} (k(I') \cdot_K k'(I'')) +_K (k(I') \cdot_K k''(I''))$$
$$= \sum_{\forall I', I'': I = I' \cap I''} (k(I') \cdot_K k'(I''))$$
$$+ \sum_{\forall I', I'': I = I' \cap I''} (k(I') \cdot_K k''(I''))$$
$$= ((k \cdot_{K_{\mathcal{P}}} k') +_{K_{\mathcal{P}}} (k \cdot_{K_{\mathcal{P}}} k''))(I)$$

$$k \cdot_{K_{\mathcal{T}}} (k' +_{K_{\mathcal{T}}} k'')$$
$$= \mathcal{C}_K(k \cdot_{K_{\mathcal{P}}} \mathcal{C}_K(k' +_{K_{\mathcal{P}}} k''))$$
$$= \mathcal{C}_K(k \cdot_{K_{\mathcal{P}}} (k' +_{K_{\mathcal{P}}} k''))$$
$$= \mathcal{C}_K((k \cdot_{K_{\mathcal{P}}} k') +_{K_{\mathcal{P}}} (k \cdot_{K_{\mathcal{P}}} k'')))$$
$$= \mathcal{C}_K(\mathcal{C}_K(k \cdot_{K_{\mathcal{P}}} k') +_{K_{\mathcal{P}}} \mathcal{C}_K(k \cdot_{K_{\mathcal{P}}} k'')))$$
$$= (k \cdot_{K_{\mathcal{T}}} k') +_{K_{\mathcal{T}}} (k \cdot_{K_{\mathcal{T}}} k'') \qquad \square$$

Proof of Theorem 6.3. Proven by substitution of definitions:
Preserves neutral elements:

$$\mathcal{C}_K(\tau_T(0_{K_{\mathcal{T}}})) = \sum_{I \in \mathbb{I}: T \in I} 0_{K_{\mathcal{T}}}(I) = \sum_{I \in \mathbb{I}: T \in I} 0_K = 0_K$$

$$\tau_T(1_{K_{\mathcal{T}}}) = \sum_{I \in \mathbb{I}: T \in I} 1_{K_{\mathcal{T}}}(T)$$

Since $T \in [T_{min}, T_{max})$ for any $T \in \mathbb{T}$ and $1_{K_{\mathcal{T}}}(I) = 0_K$ for any interval I except for $[T_{min}, T_{max})$ where $1_{K_{\mathcal{T}}}([T_{min}, T_{max})) = 1_K$ we get $\sum_{I \in \mathbb{I}: T \in I} 1_{K_{\mathcal{T}}}(T) = 1_K$
Commutes with addition:

$$\tau_T(k +_{\mathcal{T}} k') = \sum_{I \in \mathbb{I}: T \in I} (k +_{\mathcal{T}} k')(I) = \sum_{I \in \mathbb{I}: T \in I} k(I) +_K k'(I)$$

$$= \sum_{I \in \mathbb{I}: T \in I} k(I) +_K \sum_{I \in \mathbb{I}: T \in I} k'(I) = \tau_T(k) + \tau_T(k')$$

Commutes with multiplication:

$$\tau_T(k \cdot_{\mathcal{T}} k') = \sum_{I \in \mathbb{I}: T \in I} (k \cdot_{\mathcal{T}} k')(I)$$
$$= \sum_{I \in \mathbb{I}: T \in I} \sum_{\forall I', I'': I = I' \cap I''} k(I') \cdot_K k'(I'')$$
$$= \sum_{\forall I', I'': T \in I' \wedge T \in I''} k(I') \cdot_K k'(I'')$$

Let $n_1, \ldots, n_l$ denote the elements $k(I)$ for all intervals from the set of intervals with $T \in I$ and $k(I) \neq 0$. Analog, let $m_1, \ldots, m_o$ bet the set of elements with the same property for k'. Then the sum can be rewritten as:

$$= \sum_{i=1}^{l} \sum_{j=1}^{o} n_i \cdot_K m_j = \sum_{i=1}^{l} n_i \cdot_K (\sum_{j=1}^{o} m_j) = (\sum_{i=1}^{l} n_i) \cdot_K (\sum_{j=1}^{o} m_j)$$

replacing this again with the interval notation we get:

$$= (\sum_{\forall I: T \in I} k(I)) \cdot_K (\sum_{\forall I: T \in I} k'(I)) = \tau_T(k) \cdot_K \tau_T(k') \qquad \square$$

Proof of Theorem 6.4. injective: We have to show that for any two snapshot K-relations R and R', $\textsc{Enc}_K(R) = \textsc{Enc}_K(R') \Rightarrow R = R'$. Since, $\mathcal{C}_K$ preserves snapshot equivalence and is a unique representation of any temporal K-element $\mathcal{T}$, it is sufficient to show that for all t, we have $\mathcal{T}_{R,t} = \mathcal{T}_{R',t}$ instead. For sake of contradiction, assume that there exists a tuple t such that $\mathcal{T}_{R,t} \neq \mathcal{T}_{R',t}$. Then there has to exist $T \in \mathbb{T}$ such that $\mathcal{T}_{R,t}([T, T+1)) \neq \mathcal{T}_{R',t}([T, T+1))$. However, based on the definition of $\mathcal{T}_{R,t}$ this implies that $R(T)(t) \neq R'(T)(t)$ which contradicts the assumption.
surjective: Given a $K_{\mathcal{T}}$-relation R, we construct a snapshot K-relation R' such that $\textsc{Enc}_K(R') = R$: $R'(T)(t) = \sum_{T \in I} R(t)(I)$. $\square$

Proof of Lemma 6.5. By virtue of snapshot equivalence between $\mathcal{C}_K(\mathcal{T})$ and $\mathcal{T}$ and based on the singleton interval definition of $\mathcal{T}_{R,t}$ in $\textsc{Enc}_K$, we have for any tuple t:

$$\tau_T(\textsc{Enc}_K(R))(t) = \tau_T(\mathcal{T}_{R,t}) = \mathcal{T}_{R,t}([T, T+1)) = R(T)(t)$$
$$= \tau_T(R)(t) \qquad \square$$

Proof of Theorem 6.6. We have to show that $(\mathcal{DB}_{K_{\mathcal{T}}}, \textsc{Enc}_K{}^{-1}, \tau)$ fulfills conditions (1), (2), and (3) of Definition 4.6 to prove that this triple is a representation system for K-relations. Conditions (1) and (2) have been proven in Lemmas 6.4 and 6.5, respectively. Condition (3) follows from the fact that τ_T is a homomorphism (Theorem 6.3) and that semiring homomorphisms commute with $\mathcal{RA}^+$-queries ([21], Proposition 3.5). $\square$

Proof of Theorem 7.1. To prove that $K_{\mathcal{T}}$ has a well-defined monus, we have to show $K_{\mathcal{T}}$ is naturally ordered and that for any k and k', the set $\{k'' \mid k \preceq_{K_{\mathcal{T}}} k' + k''\}$ has a unique smallest element according to $\preceq_{K_{\mathcal{T}}}$. A semiring is a naturally ordered if $\preceq_{K_{\mathcal{T}}}$ is a partial order (reflexive, antisymmetric, and transitive). $k \preceq_{K_{\mathcal{T}}} k' \Leftrightarrow \exists k'' : k +_{K_{\mathcal{T}}} k'' = k'$. Substituting the definition of addition, we get $\exists k'' : \mathcal{C}_K(k +_{K_{\mathcal{P}}} k'') = k'$. Since $\mathcal{C}_K(k') = k'$ and coalesce preserves snapshot equivalence, we have $\mathcal{C}_K(k +_{K_{\mathcal{P}}} k'') = k' \Leftrightarrow \forall T \in \mathbb{T} : \tau_T(k) +_K \tau_T(k'') = \tau_T(k')$. From $\mathcal{C}_K(k +_{K_{\mathcal{P}}} k'') = k' \Leftrightarrow \forall T \in \mathbb{T} : \tau_T(k) +_K \tau_T(k'') = \tau_T(k')$ follows that $k \preceq_{K_{\mathcal{T}}} k' \Leftrightarrow \forall T \in \mathbb{T} : \tau_T(k) \preceq_K \tau_T(k')$.

Note that we only have to prove that $\preceq_{K_{\mathcal{T}}}$ is antisymmetric, since reflexivity and transitivity of the natural order follows from the semiring axioms and, thus, holds for all semirings.
Antisymmetric: We have to show that $\forall k, k' \in K_{\mathcal{T}} : k \preceq_{K_{\mathcal{T}}} k' \wedge k' \preceq_{K_{\mathcal{T}}} k \rightarrow k = k'$. This holds because, $k \preceq_{K_{\mathcal{T}}} k'$ and $k' \preceq_{K_{\mathcal{T}}} k$ iff for all $T \in \mathbb{T}$ we have $\tau_T(k) \preceq_K \tau_T(k')$ and $\tau_T(k') \preceq_K \tau_T(k)$ which implies $\tau_T(k) = \tau_T(k')$ for all $T \in \mathbb{T}$ which can only be the case if $k \sim k'$. Since k and k' are coalesced it follows that $k = k'$.
Unique Smallest Element Exists: It remains to be shown that $\{k'' \mid k \preceq_{K_{\mathcal{T}}} k' + k''\}$ has a smallest member for all $k, k' \in K_{\mathcal{T}}$. We

give a constructive proof by constructing the smallest such element k_{min}. k_{min} is defined by coalescing an element k_{pmin} that consists of singleton intervals ($[T, T+1)$) as follows:

$$k_{min} = \mathcal{C}_K(k_{pmin})$$

$$\forall I \in \mathbb{I} : k_{pmin}(I) = \begin{cases} \tau_T(k) -_K \tau_T(k') & \text{if } I = [T, T+1) \\ 0_K & \text{else} \end{cases}$$

First we have to demonstrate that indeed $k \preceq_{K_\mathcal{T}} k' +_K k_{min}$. Recall that $\mathcal{C}_K(k' +_{K_\mathcal{P}} k_{min}) = k \Leftrightarrow \forall T \in \mathbb{T} : \tau_T(k') + \tau_T(k_{min})) = \tau_T(k')$. Substituting the definition of k_{min} and using the fact that τ_T commutes with addition, for every time point T we distinguish two cases. Either $\tau_T(k') \succeq_K \tau_T(k)$ in which case $\tau_T(k) -_K \tau_T(k') = 0_K$ and we have: $\tau_T(k') +_K (\tau_T(k) -_K \tau_T(k')) = \tau_T(k') + 0_K = \tau_T(k')$. Thus, $\tau_T(k') +_K (\tau_T(k) -_K \tau_T(k')) \succeq_K k\tau_T(k')$ reduces to $\tau_T(k') \succeq_K \tau_T(k)$ which was assumed to hold.

Otherwise for $\tau_T(k') \preceq_K \tau_T(k)$ we have: $\tau_T(k') +_K (\tau_T(k) -_K \tau_T(k'))$. Let $k'' = (\tau_T(k) -_K \tau_T(k'))$. Substituting the definition of $-_K$, we get $k'' = min_{k'''} \tau_T(k') + k''' \succeq_K \tau_T(k)$. Thus,

$$\tau_T(k') +_K k'' \succeq_K k.$$

It remains to be shown that k_{min} is minimal. For contradiction assume that there exists a smaller such member k_{alt}. Then there has to exist at least one time point T such that $\tau_T(k_{alt}) \prec_K \tau_T(k_{min})$. We have to distinguish two cases. If $\tau_T(k) \preceq_K \tau_T(k')$, then $\tau_T(k_{min}) = 0_K$. However, since $0_K \leq k$ for any $k \in K$ this leads to a contradiction. Otherwise $\tau_T(k') + \tau_T(k_{alt}) \prec_K \tau_T(k') +_K \tau_T(k_{min}) = \tau_T(k)$ contradicting the assumption that $k \preceq_K k' +_K k_{alt}$. □

Proof of Theorem 7.2. We have to prove that $\tau_T(k -_{K_\mathcal{T}} k') = \tau_T(k) -_K \tau_T(k')$. We start with $\tau_T(k -_{K_\mathcal{T}} k') = \tau_T(\mathcal{C}_K(k -_{K_\mathcal{P}} k'))$. Since $\mathcal{C}_K$ preserves $\sim$ and $\tau_T(k) = \tau_T(k')$ if $k \sim k'$, we get:

$$= \sum_{T \in I} (k -_{K_\mathcal{P}} k')(I) = \qquad \tau_T(k) -_K \tau_T(k') \qquad \square$$

Proof of Theorem 7.3. By construction, the result of aggregation is a $\mathbb{N}_\mathcal{T}$ relation (it is coalesced). Also by construction, we have $\tau_T({}_G\gamma_{f(A)}(R)) = {}_G\gamma_{f(A)}(\tau_T(R))$. □

Proof of Theorem 8.1. To prove the relationships in the commutative diagram of Equation (1), we have to prove that $\text{PeriodEnc}^{-1}(\text{PeriodEnc}(R)) = R$ and that queries commute with PeriodEnc if rewritten using Rewr, i.e., $\text{PeriodEnc}(Q(R)) = \text{Rewr}(Q)(\text{PeriodEnc}(R))$.

$\text{PeriodEnc}^{-1}(\text{PeriodEnc}(R)) = R$: Let R be a $\mathbb{N}_\mathcal{T}$-relation and R' denote $\text{PeriodEnc}(R)$. Consider an arbitrary tuple t and let $\mathcal{T}$ denote the temporal element associated with t, i.e., $R(t) = \mathcal{T}$. Consider any interval $I \in \mathbb{I}$ and let $n_I = \mathcal{T}(I)$ (the multiplicity assigned by $\mathcal{T}$ to I). According to Definition 8.1, this implies that tuple $t_I = (t, I^+, I^-)$ is annotated with n_I. Let $\mathcal{T}_t$ denote the temporal element assigned by PeriodEnc^{-1} to t. By construction $\mathcal{T}_t(I) = n_I = \mathcal{T}(I)$.

$\text{PeriodEnc}(Q(R)) = \text{Rewr}(Q)(\text{PeriodEnc}(R))$: We prove this part by induction over the structure of a query. Let $R' = \text{PeriodEnc}(R)$.

Base case: Assume that $Q = R$ for some relation R. The claim follows immediately from $\text{Rewr}(R) = R$.

Induction Step: Assume the claim holds for queries with up to n operators. We have to prove the claim for any query Q with $n+1$ operators. For unary operators, WLOG let $Q = op(Q_n)$ for an operator op and query Q_n with n operators and let $Q' = \text{Rewr}(Q)$.

Selection: $op = \sigma_\theta$: A selection is rewritten as $Q' = \mathcal{C}(\sigma_\theta(\text{Rewr}(Q)))$. Consider an input tuple t from R. The temporal K-element $\mathcal{T}$ annotating tuple t is represented as a set of tuples of the form (t, I^+, I^-) for some interval I. If t fulfills the selection, then t is annotated with $\mathcal{T}$ in the result. In R', all of these tuples are in the result of Q' if $t \models \theta$ and applying PeriodEnc^{-1} we get $\mathcal{T}$ as the annotation of t. If t does not fulfill the condition then t is annotated with 0 in both encodings.

Projection: $op = \Pi_A$: A projection is rewritten by adding the attributes encoding the interval associated to a tuple to the projection expressions. There will be one tuple (t, I^+, I^-) in the result for each interval I assigned a non-zero annotation in $R(u)$ for any tuple u projected on tuple t. Function PeriodEnc^{-1} creates the annotation of an output as a temporal element that maps each interval mapping to a non-zero annotation in $\text{PeriodEnc}(R)$ to that annotation. This corresponds to addition of singleton temporal elements and based on the fact that addition is associative this implies that the annotation of t in the output will be the sum of temporal elements $R(u)$ for each u projected onto t. Thus, the claim holds.

Aggregation: $op = \gamma_{f(A)}$: The rewrite for aggregation without group-by utilizes the normalization operation $\mathcal{N}$ we have defined. Note that $\mathcal{N}_\emptyset$ returns a $\mathbb{N}_\mathcal{T}$-relation S where for any pair of tuples t and t' and any pair of intervals I_1 and I_2 we have $I_1 \neq I_2 \wedge I_1 \cap I_2 \neq \emptyset \Rightarrow S(t')(I_1) = 0 \vee S(t')(I_2) = 0$. That is, all intervals with non-zero annotations from any pair of temporal elements do not overlap or are the same. From that follows that for any two time points $T_1, T_2 \in I$ for an interval I that is mapped to $n \neq 0$ in the annotation of at least one tuple S, the value of the result of aggregation is the same for the snapshots at T_1 and T_2. Thus, grouping by the interval boundaries yields the expected result with the exception of an empty snapshot. However, since a tuple $(0_f, T_{min}, T_{max})$ is added to the input, the aggregation will produce 0 (count) or **NULL** (other aggregation functions) for intervals containing only empty snapshots. This does not effect the result of the aggregation for non-empty snapshots, because 0_f is the neutral element of the aggregation function f.

Aggregation: $op = {}_G\gamma_{f(A)}$: For aggregation with group-by, normalization is applied grouping on G and no additional tuple $(0_f, T_{min}, T_{max})$ is added to the input. Since the tuples within one group are normalized, the argument we have used above for aggregation without group-by applies also to aggregation with group-by.

For binary operators WLOG let $Q = op(Q_l, Q_r)$ where the total number of operators in Q_l and Q_r is n.

Join: $op = Q_l \bowtie_\theta Q_r$: Consider a tuple t that is the result of joining tuples u and v. Let $\mathcal{T}_u$ and $\mathcal{T}_v$ be the temporal elements annotating u and v in the input, respectively. Based on the definition of the rewriting, in the result of the rewritten join there will be a tuple t, I^+, I^- annotated with $\sum_{I_u, I_v} Q_l(u) \cdot Q_r(v)$ for all intervals I_u and I_v such that $I = I_u \cap I_c$. This corresponds to the definition of multiplication (join) in $\mathbb{N}_\mathcal{T}$.

Union: $op = Q_l \cup Q_r$: Union is rewritten as a union of the rewritten inputs. For any tuple t, let $\mathcal{T}_l = Q_l(t)$ and $\mathcal{T}_r = Q_r(t)$. In the result of the union applied by Q' a tuple (t, I^+, I^-) for each interval I will be annotated with $\mathcal{T}_l(t) + \mathcal{T}_r(t)$. The result of the union is then coalesced. Applying PeriodEnc^{-1} the annotation computed for t is equivalent to $\mathcal{C}_\mathbb{N}(\mathcal{T}_l +_{K_\mathcal{P}} \mathcal{T}_r)$.

Difference: $op = Q_l - Q_r$: A difference is rewritten by applying difference to the pairwise normalized inputs. Recall that the monus operator of $\mathbb{N}_\mathcal{T}$ associates the result of the monus for $\mathbb{N}$ to each snapshot of a temporal $\mathbb{N}$-element. Since normalization adjusts intervals such that there is no overlap, the claim holds. □

B. QUERY DESCRIPTIONS

join-1. Return the salary and department for every employee.

```
SELECT a.emp_no, dept_no, salary
FROM dept_emp a JOIN salaries b ON (a.emp_no = b.emp_no)
```

join-2. Return the department, salary, and title for every employee.

```
SELECT title, salary, dept_no
FROM dept_emp a JOIN salaries b ON (a.emp_no = b.emp_no)
                JOIN titles c ON (a.emp_no = c.emp_no)
```

join-3. Return employees that manage a particular department and earn more then $70,000.

```
SELECT a.emp_no, dept_no
FROM dept_manager a
     JOIN salaries b ON (a.emp_no = b.emp_no)
WHERE salary > 70000
```

join-4. Returns information about the manager of each department.

```
SELECT a.emp_no, a.dept_no, b.salary, first_name,
       last_name
FROM dept_manager a, salaries b, employees e
WHERE a.emp_no = b.emp_no and a.emp_no = e.emp_no
```

agg-1. Returns the average salary of employees per department.

```
SELECT dept_no, avg(salary) as avg_salary
FROM dept_emp a
     JOIN salaries b ON (a.emp_no = b.emp_no)
GROUP BY dept_no
```

agg-2. Returns the average salary of managers.

```
SELECT avg(salary) as avg_salary
FROM dept_manager a
     JOIN salaries b ON (a.emp_no = b.emp_no)
```

agg-3. Returns the number of departments with more than 21 employees.

```
SELECT count(1)
FROM (SELECT count(*) AS c, dept_no
    FROM dept_emp WHERE emp_no < 10282
    GROUP BY dept_no HAVING count(*) > 21) s
```

agg-join. Returns the names of employees with the highest salary in their department. It contains a 4-way join where one of the join inputs is the result of a subquery with aggregation.

```
SELECT d.emp_no, e.first_name, e.last_name,
       maxS.max_salary, d.dept_no
FROM (SELECT max(salary) as max_salary,dept_no
      FROM dept_emp a
           JOIN salaries b ON (a.emp_no = b.emp_no)
      GROUP BY dept_no) maxS,
      salaries s, dept_emp d, employees e
WHERE e.emp_no = s.emp_no
      AND s.salary = maxS.max_salary
    AND d.dept_no = maxS.dept_no
      AND d.emp_no = e.emp_no
```

diff-1. Returns employees that are not managers of any department.

```
SELECT emp_no FROM dept_emp
EXCEPT ALL
SELECT emp_no FROM dept_manager
```

diff-2. Returns salaries of employees that are not managers.

```
SELECT a.emp_no, salary
FROM (SELECT emp_no FROM dept_emp
      EXCEPT ALL
      SELECT emp_no FROM dept_manager) a
      JOIN salaries b ON (a.emp_no = b.emp_no)
```

C-Sn. To evaluate the performance of coalescing we use the following query template varying the selection condition on salary to control the size of the output. The query returns employee salaries. We materialize the result of this query for each selectivity and use this as the input to coalescing.

```
SELECT a.EMP_NO, salary
FROM employees a
     JOIN salaries b ON (a.emp_no = b.emp_no)
WHERE salary > ?
```

C. PULLING-UP COALESCING

The main overhead of our approach for sequenced temporal queries compared to non-temporal query processing is the extensive use of coalescing, which can be expensive if naively implemented in SQL. Furthermore, the application of coalescing after each operation may prevent the database optimizer from applying standard optimizations such as join reordering. To address this issue, we now investigate how to reduce the number of coalescing steps. In fact, we demonstrate that it is sufficient to apply coalescing as a last step in query processing instead of applying it to intermediate results. Similar optimizations have been proposed by Bowman et al. [11] for their multiset temporal normalization operator and by Böhlen et al. [9] for set-coalescing.

Consider how a $\mathcal{RA}^+$ query Q is evaluated over an $K_\mathcal{T}$-database. $\mathcal{RA}^+$ over K-relations computes the annotation of a tuple in the result of a query using the addition and multiplication operations of the semiring. That is, the annotation of any result tuple is computed using an arithmetic expression over the annotations of tuples from the input of the query. In the case of a semiring $K_\mathcal{T}$, addition and multiplication are defined as coalescing a temporal element that is computed based on point-wise application of the addition (multiplication) operations of semiring K (denoted as $+_{K_\mathcal{P}}$ and $\cdot_{K_\mathcal{P}}$). Recall from Lemma 6.1 that coalescing can be redundantly pushed into the addition and multiplication operations of interval-temporal semirings, e.g., $\mathcal{C}_K(k +_{K_\mathcal{P}} k') = \mathcal{C}_K(\mathcal{C}_K(k) +_{K_\mathcal{P}} k')$. Interpreting this equivalence from right to left and applying it repeatedly to an arithmetic expression e using $+_{K_\mathcal{T}}$ and $\cdot_{K_\mathcal{T}}$, the expression can be rewritten into an equivalent expression of the form $\mathcal{C}_K(e')$, where e' is an expression that only uses operations $+_{K_\mathcal{P}}$ and $\cdot_{K_\mathcal{P}}$. Now consider expressions that also include applications of the monus operator $-_{K_\mathcal{T}}$. This operator is defined as $\mathcal{C}_K(k -_{K_\mathcal{P}} k')$. The $-_{K_\mathcal{P}}$ operator computes the timeslice of the inputs at every point in time and then applies $-_K$ to each timeslice. According to Lemma 5.1, $\tau_T(k) \sim \tau_T(\mathcal{C}_K(k'))$. Thus, the result of $-_{K_\mathcal{P}}$ is independent of whether the input is coalesced or not.

Lemma C.1. *Any arithmetic expression e using operations and elements from an period m-semiring $K_\mathcal{T}$ is equivalent to an expression of the form $\mathcal{C}_K(e')$, where e' only contains operations $+_{K_\mathcal{P}}$, $\cdot_{K_\mathcal{P}}$, and $-_{K_\mathcal{P}}$.*

Lemma C.1 implies that it is sufficient to apply coalescing as a last step in a rewritten query $\text{REWR}(Q)$ instead of after each operator.

Corollary C.2 (Coalesce Pullup)**.** *For any $\mathcal{RA}$ query Q, $\text{REWR}(Q)$ is equivalent to a query Q' which is derived from $\text{REWR}(Q)$ by removing all but the outermost coalescing operator.*

Proof. Operations $+_{K_\mathcal{T}}$, $\cdot_{K_\mathcal{T}}$, and $-_{K_\mathcal{T}}$ are defined as applying $\mathcal{C}_K$ to the result of operations $+_{K_\mathcal{P}}$, $\cdot_{K_\mathcal{P}}$, and $-_{K_\mathcal{P}}$, respectively. Thus, expression e is equivalent to an expression that interleaves the $\mathcal{C}_K$ as well as $+_{K_\mathcal{P}}$, $-_{K_\mathcal{P}}$, and $\cdot_{K_\mathcal{P}}$ operations. To prove this, we first prove that the following equivalence holds: $\mathcal{C}_K(k -_{K_\mathcal{P}} k') \Leftrightarrow \mathcal{C}_K(\mathcal{C}_K(k) -_{K_\mathcal{P}} k') \Leftrightarrow \mathcal{C}_K(k -_{K_\mathcal{P}} \mathcal{C}_K(k'))$. Consider the definition of $-_{K_\mathcal{P}}$. Every interval $I = [T, T+1)$ is assigned the annotation $\tau_T(k) -_K \tau_T(k')$. Applying Lemma 5.1 we get $\tau_T(k) = \tau_T(\mathcal{C}_K(k))$ and $\tau_T(k') = \tau_T(\mathcal{C}_K(k'))$. Thus, the equivalence holds. By repeatedly applying this equivalence and the equivalences proven in Lemma 6.1, all except the outermost K-coalesce operations can be removed resulting in an expression of the form $\mathcal{C}_K(e')$ where e' does not contain any coalesce operations. □

Example C.1. *Consider the following query $Q = S - \Pi_{sal}(\sigma_{sal<sal'}(S \times \rho_{sal' \leftarrow sal}(S))$ that returns the largest salary from relation S as shown in Figure 3 (consider the corresponding $\mathbb{N}_\mathcal{T}$-relation using the annotation shown on the right in this figure coalesced as shown in Example 5.3). Consider how the annotation of tuple $r = (50k)$ in the result of Q is computed. Applying the definitions of difference, projection, and join over K-relations and denoting the database instance of S as D, we obtain:*

$$Q(D)(t) = \mathcal{C}_\mathbb{N}(S(t) -_{K_\mathcal{P}} \mathcal{C}_\mathbb{N}(\sum_{u=(v,w):u.sal=t} \mathcal{C}_\mathbb{N}(\mathcal{C}_\mathbb{N}((S(v) \cdot_{K_\mathcal{P}} S(w))) \cdot_{K_\mathcal{P}} (sal < sal')(u))))$$

Pulling up coalesce we get:

$$Q(D)(t) = \mathcal{C}_\mathbb{N}(S(t) -_{K_\mathcal{P}} \sum_{u=(v,w):u.sal=t} (S(v) \cdot_{K_\mathcal{P}} S(w)) \cdot_{K_\mathcal{P}} (sal < sal')(u))$$