

Querying Interval-Based Temporal Data with SPARQL

Julien Corman ✉ 

Free University of Bozen-Bolzano, Italy

Roman Kontchakov ✉ 

Birkbeck, University of London, UK

Cem Okulmus ✉ 

Paderborn University, Germany

Abstract

We propose an extension of the graph query language SPARQL to support temporal data. The proposal follows the approach of temporal databases, where each tuple is annotated with a finite set of intervals, and joins and other operators implicitly take account of these annotations. We adapt this model to the specifics of SPARQL query evaluation, which is defined in terms of compatibility of partial solution mappings. Furthermore, unlike existing proposals, our model allows predicates over intervals (e.g., Allen's relations) in temporal atoms of filter expressions, while ensuring that query evaluation remains tractable in data complexity. Our semantics comes in three equivalent flavours, with different levels of abstraction. The basic semantics provides a snapshot view of data (it is, however, applicable only to queries without temporal atoms). In the second semantics, induced by an SPM-semiring, solution mappings are annotated with finite sets of intervals. This semantics covers temporal atoms and coincides with the snapshot semantics when the latter is defined. The third, most concrete semantics is equivalent to the second, but based on single intervals, which naturally leads to an implementation over a triple store or a relational database, with tables featuring columns for interval endpoints. As a proof of concept, we evaluate a database storage prototype and show that it addresses previously identified performance limitations of triple stores for temporal data.

2012 ACM Subject Classification Information systems → Temporal data; Information systems → Query languages

Keywords and phrases Query languages, SPARQL, interval-based temporal data

Digital Object Identifier 10.4230/OASICS.TIME.2026.6

1 Introduction

Temporal data has long been of interest to various fields of computer science, from database theory to knowledge representation, and is also seeing growing applications, as evidenced by the continued interest in specialised database systems dealing with time, such as time-series databases and extensions of the SQL standard. Notably, SQL:2011 features time period definitions, temporal integrity constraints and temporal predicates (Allen's relations).

Formal models of databases (in particular, temporal databases) are typically grounded in relational algebra, which makes strong assumptions about the structure of the data being queried. As a result, these models do not easily extend to missing, ill-typed or heterogeneous data. In contrast, knowledge graphs (KGs) and associated query languages provide a robust framework for modelling such data. However, temporal KGs remain largely underdeveloped.

In this work, we focus on the graph query language SPARQL, a widely used W3C standard. At first sight, SPARQL is similar to SQL, but its core algebra operations (union, join and difference) are of a different nature: they operate on (multi)sets of *partial* functions rather than relations, which can be regarded as (multi)sets of *total* functions. This distinction has far-reaching consequences when it comes to equipping SPARQL with temporal features,



© Julien Corman, Roman Kontchakov, and Cem Okulmus;
licensed under Creative Commons License CC-BY 4.0

33rd International Symposium on Temporal Representation and Reasoning (TIME 2026).

Editors: AndreA Orlandini and Sophie Pinchinat; Article No. 6; pp. 6:1–6:21

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Listing 1** An example Temporal SPARQL query, adapted from [14].

```
PREFIX : <http://www.example.org#>
PREFIX time: <http://www.w3.org/2006/time#>
SELECT ?e ?dep1 ?t1 {
  << ?e      :worksIn      ?dep1      >> time:hasTime ?t1 .
  << ?dep1 :locatedIn :barcelona >> time:hasTime ?t1 .
  FILTER NOT EXISTS {
    << ?e      :worksIn      ?dep2 >> time:hasTime ?t2 .
    << ?dep2 :locatedIn :rome >> time:hasTime ?t2 .
    FILTER ( time:Before(?t2,?t1) ).
  }
}
```

and, to our knowledge, proposals in this direction have largely ignored it. In other words, some foundational work is still missing that provides SPARQL with temporal abilities while respecting its semantics, which is the gap that we propose to fill.

More precisely, we adapt to SPARQL the classical model of temporal relational databases, where data and answers are annotated with intervals, and extend it with the possibility to compare maximal temporal intervals. As a concrete illustration, we reproduce an example by Klarman and Meyer [14].¹

► **Example 1.** Consider the query in Listing 1, given in SPARQL-star syntax, where reified triples are enclosed in `<<...>>`. The query retrieves employees `?e` who worked in department `?dep1` located in Barcelona during some time interval `?t1`, but did not work in any department `?dep2` located in Rome during an interval `?t2` that immediately precedes `?t1` (Allen’s relation `time:Before`). Each answer consists of one such employee, with their department `?dep1` and the relevant interval `?t1`. Observe that, similarly to temporal databases [2], all the joins in the query are (*implicitly*) *temporal*. For instance, the temporal variable `?t1` in the query result represents an interval when the following two statements annotated with `time:hasTime ?t1` hold: (i) `?e` worked in `?dep1` and (ii) `?dep1` was based in Barcelona. However, such an interval may not come directly from the data: it is instead the intersection of two *maximal* intervals where (i) and (ii) respectively hold (for the same employee and department). In other words, `?t1` ranges over maximal periods of time where both statements hold, and similarly for `?t2`. This ensures that the temporal atom `time:Before(?t2,?t1)` can be interpreted unambiguously.

As this example illustrates, preserving maximality of temporal intervals, which is called *temporal coalescing* [3], is crucial for a clear and well-defined semantics of *temporal atoms*, which feature predicates on intervals, for instance, Allen’s relations. On the other hand, as we shall see below, an unrestricted use of temporal variables could lead to either intractability (in the size of the data) or a semantics that is not well-defined. Careful restrictions on the *scope* of temporal variables are thus needed, and our proposal achieves this by means of SPARQL’s EXISTS operator, as illustrated in Listing 1.

Our main contributions are as follows.

¹ Note, however, that Klarman and Meyer employ two-sorted first-order query language and consider background knowledge (ontologies) for enriching query answering, whereas our focus is solely on SPARQL and its algebra, without any ontologies.

1. We identify a suitable fragment of SPARQL-star (an extension of SPARQL, originally proposed by Hartig and Thompson [11] and is standardised by the W3C as part of the upcoming SPARQL 1.2) to easily express temporal join operations as well as comparison of intervals (using temporal atoms with Allen’s relations, for example). We impose suitable restrictions on this fragment to ensure that it admits a well-defined semantics and tractable query evaluation. We dub this language Temporal SPARQL.
2. We provide a formal semantics for Temporal SPARQL that complies with the specifics of the SPARQL algebra. More precisely, we follow the approach of Dignös et al. [7] to temporal databases and show that this semantics is induced by an annotation semiring suitable for SPARQL [9].
3. We conclude the paper by a brief feasibility study, which explores how Temporal SPARQL can be effectively translated into SQL. Among other applications, this makes Temporal SPARQL a good candidate input language for temporal Ontology-Based Data Access (OBDA) [12], which would bring the original proposal of Klarman and Meyer [14] closer to an effective and standard-compliant implementation. Integration with ontologies and query rewriting, however, is left for future work.

Related Work. Chekol et al. [4] present algorithms for joins over interval-based temporal data as well as an efficient algorithm for performing temporal coalescing and provide a proof-of-concept implementation to show how their proposal outperforms direct translations into standard SPARQL running on existing triple stores. Geerts et al. [9] describe the semantics of SPARQL based on algebraic structures called SPM-semirings with the aim of capturing provenance of answers to SPARQL queries on RDF graphs annotated with elements from some domain K . Dignös et al. [7] propose a semantics for snapshot-reducible database queries over relational interval-based temporal data called K -relations. While not directly applicable to our model, which uses SPARQL semantics as a starting point, we follow their presentation whereby the semantics is presented as three distinct “models” of increasing abstraction: the implementation model relies on the structure of the database tables directly, while the logical and abstract models use algebraic structures that enable easier formal reasoning. Kaminski et al. [13] extend the algebraic description of the semantics of core SPARQL [15] to the key features of SPARQL 1.1 including aggregation and nested queries with the EXISTS operator. Our definitions build on the context-based interpretation of the semantics of EXISTS by Kaminski et al. [13]. In the wider literature on temporal query languages, we also note the influential work of Böhlen et al. [3] on temporal coalescing and bespoke algorithms for its implementation. Later extensions of the SQL standard introduced window functions, which are used by Zhou et al. [20] for a more efficient translation of temporal coalescing into SQL. Dignös et al. [6] also provide examples of efficient implementation of temporal inner joins, left joins and temporal coalescing in SQL with window functions.

The rest of the paper is structured as follows. In Section 2, we introduce the syntax of Temporal SPARQL and initially focus on its small fragment (with a single temporal variable and without temporal atoms), which has a simple *snapshot* semantics. In Section 2.3, we switch to the *logical model*, in which query answers are annotated with finite set of intervals and show that the semantics with annotations from a particular SPM-semiring matches the snapshot semantics for this small fragment. In Section 3, we return to Temporal SPARQL with multiple temporal variables and temporal atoms, and introduce syntactic restrictions that ensure tractable query answering in the logical model. In Section 4, we define the so-called *concrete model*, in which answers are assigned single intervals and which is suitable for a translation into the relational setting. In Section 5, we provide a feasibility study that shows how Temporal SPARQL can be effectively and efficiently translated to SQL.

2 Temporal SPARQL: Syntax and Semantics

In Section 2.1, we provide an abstract syntax for Temporal SPARQL, broadly following [18], which is in turn based on the simplified abstract SPARQL algebra from the W3C recommendation². This abstract syntax has a natural SPARQL-star counterpart, where reified triples are annotated with temporal variables, as illustrated in Example 1; note, however, that in Section 3.1, we will need to introduce additional syntactic restrictions to ensure that query evaluation is tractable in data complexity. But first, in Section 2.2, we study an even smaller fragment of Temporal SPARQL that uses a single temporal variable and define its *snapshot* semantics, which coincides with the standard (non-temporal) SPARQL semantics when the timepoint of query evaluation is fixed. In Section 2.3, we equip this snapshot fragment with an equivalent annotation-based semantics based on SPM-semirings, which generalises a wide range of semantics for SPARQL (sets, bags, provenance, etc.).

2.1 Syntax

We assume pairwise disjoint infinitely countable sets of *IRIs* $\mathcal{I}$, *blank nodes* $\mathcal{B}$, *literals* $\mathcal{L}$, *domain variables* $\mathcal{V}$ and *temporal variables* $\mathcal{T}$. The *domain* $\mathcal{D}$ is the union $\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$. *Graph patterns* P and *filter expressions* F are defined by mutual recursion as follows:

$$\begin{aligned} P &:= (s, p, o, t) \mid \text{FILTER}(P, F) \mid \text{JOIN}(P, P) \mid \text{OPTIONAL}(P, P, F) \mid \text{UNION}(P, P) \\ &\quad \mid \text{MINUS}(P, P) \mid \text{PROJECT}(P, L) \mid \text{EXTEND}(P, v, T), \\ F &:= \text{BOUND}(v) \mid R(T_1, \dots, T_n) \mid A(t_1, \dots, t_n) \mid \text{EXISTS}(P) \mid F \wedge F \mid \neg F, \end{aligned}$$

where $s, p, o \in \mathcal{V} \cup \mathcal{D}$ are (*basic*) *domain terms*, $v \in \mathcal{V}$ and $t, t_1, \dots, t_n \in \mathcal{T}$ are domain and temporal variables, respectively, $L \subseteq \mathcal{V}$ a finite set of domain variables, R and A are n -ary relations over domain elements and time intervals, respectively, and $T, T_1, \dots, T_n$ are *domain terms* built from elements of $\mathcal{V}$ and $\mathcal{D}$ using domain functions such as $+$, CONCAT , etc. The basic building block of graph patterns, (s, p, o, t) , is called a *temporal quad pattern*: s is the *subject*, p is the *predicate*, o is the *object* and t is the temporal variable of the statement. In the concrete SPARQL-star syntax, illustrated in Example 1, (s, p, o) forms a reified triple, used as the subject of a triple pattern with predicate `time:hasTime` and object t . As usual in SPARQL,³ we assume that each variable v in $\text{EXTEND}(P, v, T)$ does not occur in pattern P .

2.2 Abstract Model: Snapshot Semantics

We first consider a fragment of Temporal SPARQL reducible to the snapshot view of temporal data. A graph pattern is called *snapshot* if it uses a single temporal variable, t , and does not contain filter (sub-)expressions of the form $A(t_1, \dots, t_n)$ or $\text{EXISTS}(P)$. Note that the query in Listing 1 is not snapshot, as it contains two temporal variables, `t1` and `t2`. In contrast, the queries in Listing 2 in Section 5.1 and in Listings 4 and 5 in Section 5.2 are all snapshot.

A *temporal domain* $\mathbb{T}$ is a linearly ordered set, say, $\mathbb{Q}$ or $\mathbb{N}$. The strict linear order on $\mathbb{T}$ will be denoted by $<$, with $\leq$ and other relations defined in the usual way. We denote by $\mathcal{RDF}$ the set $(\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times \mathcal{D}$, whose elements are called *RDF triples*.⁴ An (atemporal) *RDF graph* is a finite subset of $\mathcal{RDF}$. A *snapshot RDF graph* $\mathcal{G}$ maps each time instant $\tau \in \mathbb{T}$

² <https://www.w3.org/TR/sparql11-query/#sparqlAlgebra>

³ <https://www.w3.org/TR/sparql11-query/#bind>

⁴ According to the RDF 1.1 W3C recommendation (<https://www.w3.org/TR/rdf11-concepts/#section-triples>), a literal can only occur as an object, and a predicate can only be an IRI, but this is not essential for us.

to an RDF graph $\mathcal{G}(\tau)$. The *snapshot semantics* of snapshot graph patterns closely follows the standard set semantics of SPARQL and boils down to evaluation on a single $\mathcal{G}(\tau)$. We reproduce it here so that it can be compared to our interval-based semantics.

A (non-temporal) *solution mapping* μ is a partial map from $\mathcal{V}$ to $\mathcal{D}$ with a finite domain, which is denoted by $\text{dom}(\mu)$. We say that two solution mappings μ and μ' are *compatible* and write $\mu \sim \mu'$ if $\mu(v) = \mu'(v)$, for each $v \in \text{dom}(\mu) \cap \text{dom}(\mu')$. In case $\mu \sim \mu'$, we write $\mu \dot{\cup} \mu'$ for the solution mapping such that $\text{dom}(\mu \dot{\cup} \mu') = \text{dom}(\mu) \cup \text{dom}(\mu')$ and $(\mu \dot{\cup} \mu')(v) = \mu(v)$ for all $v \in \text{dom}(\mu)$ and $(\mu \dot{\cup} \mu')(v) = \mu'(v)$ for all $v \in \text{dom}(\mu')$. We denote by $\mu|_L$ the restriction of μ on L , that is, the solution mapping with domain $\text{dom}(\mu) \cap L$ that is compatible with μ .

The semantics of a domain term T is defined as usual: given a solution mapping μ , set

$$\begin{aligned} \llbracket c \rrbracket^\mu &= c, & \llbracket v \rrbracket^\mu &= \begin{cases} \mu(v), & \text{if } v \in \text{dom}(\mu), \\ \epsilon, & \text{otherwise,} \end{cases} \\ \llbracket f(T_1, \dots, T_n) \rrbracket^\mu &= \begin{cases} f(\llbracket T_1 \rrbracket^\mu, \dots, \llbracket T_n \rrbracket^\mu), & \text{if } \llbracket T_1 \rrbracket^\mu \neq \epsilon, \dots, \llbracket T_n \rrbracket^\mu \neq \epsilon, \\ \epsilon, & \text{otherwise,} \end{cases} \end{aligned}$$

where ϵ denotes the *error value*, which is used in SPARQL to deal with the fact that solution mappings are partial functions. Next, for a solution mapping μ , the following definitions reflect the standard 3-valued logic of SPARQL filter expressions F (see, e.g., [18, Section 2.1]), where the subscript s stands for “snapshot” semantics:

$$\begin{aligned} \llbracket \text{BOUND}(v) \rrbracket_s^\mu &= \begin{cases} \top & \text{if } v \in \text{dom}(\mu), \\ \perp & \text{otherwise,} \end{cases} & \llbracket R(T_1, \dots, T_n) \rrbracket_s^\mu &= \begin{cases} \epsilon & \text{if } \llbracket T_i \rrbracket^\mu = \epsilon, \text{ for some } i \leq n, \\ \top & \text{if } R(\llbracket T_1 \rrbracket^\mu, \dots, \llbracket T_n \rrbracket^\mu), \\ \perp & \text{otherwise,} \end{cases} \\ \llbracket \neg F \rrbracket_s^\mu &= \begin{cases} \epsilon & \text{if } \llbracket F \rrbracket_s^\mu = \epsilon, \\ \top & \text{if } \llbracket F \rrbracket_s^\mu = \perp, \\ \perp & \text{otherwise,} \end{cases} & \llbracket F \wedge F' \rrbracket_s^\mu &= \begin{cases} \epsilon & \text{if } \llbracket F \rrbracket_s^\mu = \epsilon \text{ or } \llbracket F' \rrbracket_s^\mu = \epsilon, \\ \top & \text{if } \llbracket F \rrbracket_s^\mu = \llbracket F' \rrbracket_s^\mu = \top, \\ \perp & \text{otherwise.} \end{cases} \end{aligned}$$

Next, let G denote the snapshot RDF graph $\mathcal{G}$ at time τ , that is, $\mathcal{G}(\tau)$. The *evaluation* $\llbracket P \rrbracket_s^G$ of a snapshot graph pattern P over G is defined by recursion on the structure of P as usual (see, e.g., [18, Section 2.1]):

$$\begin{aligned} \llbracket (s, p, o, t) \rrbracket_s^G &= \{ \mu \mid \text{dom}(\mu) = \text{var}(s, p, o) \text{ and } (s, p, o)^\mu \in G \}, \\ \llbracket \text{FILTER}(P_1, F) \rrbracket_s^G &= \{ \mu \in \llbracket P_1 \rrbracket_s^G \mid \llbracket F \rrbracket_s^\mu = \top \}, \\ \llbracket \text{JOIN}(P_1, P_2) \rrbracket_s^G &= \{ \mu_1 \dot{\cup} \mu_2 \mid \mu_1 \in \llbracket P_1 \rrbracket_s^G \text{ and } \mu_2 \in \llbracket P_2 \rrbracket_s^G \text{ with } \mu_1 \sim \mu_2 \}, \\ \llbracket \text{UNION}(P_1, P_2) \rrbracket_s^G &= \llbracket P_1 \rrbracket_s^G \cup \llbracket P_2 \rrbracket_s^G, \\ \llbracket \text{OPTIONAL}(P_1, P_2, F) \rrbracket_s^G &= \llbracket \text{FILTER}(\text{JOIN}(P_1, P_2), F) \rrbracket_s^G \\ &\quad \cup \{ \mu_1 \in \llbracket P_1 \rrbracket_s^G \mid \llbracket F \rrbracket_s^{\mu_1 \dot{\cup} \mu_2} \neq \top, \text{ for all } \mu_2 \in \llbracket P_2 \rrbracket_s^G \text{ with } \mu_1 \sim \mu_2 \}, \\ \llbracket \text{MINUS}(P_1, P_2) \rrbracket_s^G &= \{ \mu_1 \in \llbracket P_1 \rrbracket_s^G \mid \text{there is no } \mu_2 \in \llbracket P_2 \rrbracket_s^G \text{ with } \mu_1 \sim \mu_2 \\ &\quad \text{and } \text{dom}(\mu_1) \cap \text{dom}(\mu_2) \neq \emptyset \}, \\ \llbracket \text{PROJECT}(P_1, L) \rrbracket_s^G &= \{ \mu|_L \mid \mu \in \llbracket P_1 \rrbracket_s^G \}, \\ \llbracket \text{EXTEND}(P_1, v, T) \rrbracket_s^G &= \{ \mu \dot{\cup} \{v \mapsto \llbracket T \rrbracket^\mu\} \mid \mu \in \llbracket P_1 \rrbracket_s^G \}, \end{aligned}$$

where $\text{var}(s, p, o)$ denotes the set of variables in a tuple (s, p, o) of basic domain terms, and $(s, p, o)^\mu$, for a solution mapping μ , denotes the image of the tuple under μ , that is, the result of replacing each variable $v \in \text{dom}(\mu)$ in it with $\mu(v)$. Note that this semantics is applicable only to snapshot graph patterns, which have a single temporal variable, t , and therefore this variable plays no role in the definition; the time instant is fixed with the choice of $G = \mathcal{G}(\tau)$.

2.3 Logical Model: Temporal Annotations

Similarly to what was done by Dignös et al. [7] for relational algebra, we introduce an alternative representation of the semantics above, where, for a given graph, a query maps a solution mapping to (the union of) a set of intervals. This allows us to relate Temporal SPARQL to a long line of research in relational databases (notably by Green et al. [10]), and more recently for SPARQL by Geerts et al. [9], where a query maps tuples to annotations in a semiring (possibly extended with other operators [8, 9]). In this view, the semantics of a query language is defined for a whole class of semirings, thus providing a shared abstraction (as well as algorithms, proof techniques, etc.) for seemingly different formalisms, which include standard set and bag semantics, but also data annotated with provenance or probabilities. As an additional benefit, this abstract view will ease the correspondence with the more procedural model introduced in Section 4.

An *interval* over $\mathbb{T}$ is either $\mathbb{T}$ itself or one of the following subsets of $\mathbb{T}$:

- $\{\tau \in \mathbb{T} \mid \tau_1 \leq \tau < \tau_2\}$, denoted by $[\tau_1, \tau_2)$, for $\tau_1, \tau_2 \in \mathbb{T}$,
- $\{\tau \in \mathbb{T} \mid \tau_1 \leq \tau\}$, for $\tau_1 \in \mathbb{T}$, or
- $\{\tau \in \mathbb{T} \mid \tau < \tau_2\}$, for $\tau_2 \in \mathbb{T}$;

note that these intervals include the left endpoint (if any) but not the right endpoint. We use $\mathbb{I}$ to denote the set of all intervals.

We consider the set $S_{\mathbb{T}}$ of all subsets of $\mathbb{T}$ that can be represented as finite sets of intervals. Formally, $S_{\mathbb{T}} = \{\bigcup I \mid I \in \mathcal{P}_{fin}(\mathbb{I})\}$, where $\mathcal{P}_{fin}(\mathbb{I})$ is the set of finite subsets of $\mathbb{I}$. The set $S_{\mathbb{T}}$ is closed under finite set union ($\cup$) by definition. We show that it is also closed under set difference ($\setminus$), and so under finite set intersection ($\cap$), since $A \cap B = A \setminus (A \setminus B)$ for any sets A and B . To this end, we first observe that, for any $I, J \in \mathbb{I}$, the set $I \setminus J$ is the empty set, or a single element of $\mathbb{I}$, or the union of two such intervals, and so, in each case, $I \setminus J \in S_{\mathbb{T}}$. Then we can lift this to elements of $S_{\mathbb{T}}$ observing that $(I_1 \cup I_2) \setminus J = (I_1 \setminus J) \cup (I_2 \setminus J)$ and $I \setminus (J_1 \cup J_2) = (I \setminus J_1) \setminus J_2$. So, for any elements $I = \bigcup_{i=1}^n I_i$ and $J = \bigcup_{j=1}^m J_j$ of $S_{\mathbb{T}}$, with $I_i, J_j \in \mathbb{I}$ for each i and j , by induction on $n + m$, we obtain $I \setminus J \in S_{\mathbb{T}}$.

Interestingly, the algebraic structure $\mathcal{S}_{\mathbb{T}} = \langle S_{\mathbb{T}}, \cup, \cap, \setminus, \emptyset, \mathbb{T} \rangle$ belongs to the equational class (a.k.a. variety) called SPM-semirings, where “SPM” stands for “SPARQL Minus” [9]. An *SPM-semiring* $\langle S, \oplus, \otimes, \ominus, \mathbf{0}, \mathbf{1} \rangle$ is a commutative semiring (i.e., its multiplicative operation $\otimes$ is also commutative) equipped with a difference operation $\ominus$ satisfying the following identities:

$$a \ominus a = \mathbf{0}, \tag{id_1}$$

$$a \ominus (b \oplus c) = (a \ominus b) \ominus c, \tag{id_2}$$

$$a \otimes (b \ominus c) = (a \otimes b) \ominus c, \tag{id_3}$$

$$(a \ominus (a \ominus b)) \oplus (a \ominus b) = a. \tag{id_4}$$

One can easily verify that $\mathcal{S}_{\mathbb{T}}$ is an SPM-semiring. Indeed, both $\langle S_{\mathbb{T}}, \cup, \emptyset \rangle$ and $\langle S_{\mathbb{T}}, \cap, \mathbb{T} \rangle$ are commutative monoids since $\cup$ and $\cap$ are associative and commutative, and both $I \cup \emptyset = I$ and $I \cap \mathbb{T} = I$ hold for any $I \in S_{\mathbb{T}}$. Next, $\cap$ distributes over $\cup$ and $I \cap \emptyset = \emptyset$. Finally, the four identities for $\ominus$ hold for any sets A, B and C (so in particular for the elements of $S_{\mathbb{T}}$): $A \setminus A = \emptyset$, $A \setminus (B \cup C) = (A \setminus B) \setminus C$, $A \cap (B \setminus C) = (A \cap B) \setminus C$ and $(A \setminus (A \setminus B)) \cup (A \setminus B) = A$.

Geerts et al. [9] define a semantics for SPARQL queries induced by an arbitrary SPM-semiring: each triple in an RDF graph and each solution mapping in the evaluation of a query is annotated with an element of the semiring. With an appropriate choice of the semiring, this semantics coincides with the standard set or bag semantics of SPARQL (among others). We reproduce the SPM-semiring semantics of SPARQL [9], slightly extended to accommodate for OPTIONAL with a filter expression (in the original paper [9], only a tautology $\top$ is used as a filter expression in OPTIONAL). Let $\mathcal{S} = \langle S, \oplus, \otimes, \ominus, \mathbf{0}, \mathbf{1} \rangle$ be an SPM-semiring. An

$\mathcal{S}$ -annotated RDF graph is a function $G: \mathcal{RDF} \rightarrow S$ that maps finitely many triples to an element different from $\mathbf{0}$ (intuitively, $G(s, p, o) \neq \mathbf{0}$ if the RDF graph contains (s, p, o)). The evaluation $\llbracket P \rrbracket_a^G$ of a snapshot graph pattern P over an $\mathcal{S}$ -annotated RDF graph G is defined as follows (the subscript a in $\llbracket P \rrbracket_a^G$ stands for “annotation”-based semantics):

$$\begin{aligned} \llbracket (s, p, o, t) \rrbracket_a^G(\mu) &= \begin{cases} G((s, p, o)^\mu), & \text{if } \text{dom}(\mu) = \text{var}(s, p, o), (s, p, o)^\mu \in \mathcal{RDF}, \\ \mathbf{0}, & \text{otherwise,} \end{cases} \\ \llbracket \text{FILTER}(P_1, F) \rrbracket_a^G(\mu) &= \llbracket P_1 \rrbracket_a^G(\mu) \otimes \llbracket F \rrbracket_s^\mu, \\ \llbracket \text{JOIN}(P_1, P_2) \rrbracket_a^G(\mu) &= \bigoplus_{\mu_1 \dot{\cup} \mu_2 = \mu} \left(\llbracket P_1 \rrbracket_a^G(\mu_1) \otimes \llbracket P_2 \rrbracket_a^G(\mu_2) \right), \\ \llbracket \text{UNION}(P_1, P_2) \rrbracket_a^G(\mu) &= \llbracket P_1 \rrbracket_a^G(\mu) \oplus \llbracket P_2 \rrbracket_a^G(\mu), \\ \llbracket \text{OPTIONAL}(P_1, P_2, F) \rrbracket_a^G(\mu) &= \llbracket Q \rrbracket_a^G(\mu) \oplus \left(\llbracket P_1 \rrbracket_a^G(\mu) \ominus \bigoplus_{\mu' \sim \mu} \llbracket Q \rrbracket_a^G(\mu' \dot{\cup} \mu) \right), \\ &\quad \text{where } Q = \text{FILTER}(\text{JOIN}(P_1, P_2), F), \\ \llbracket \text{MINUS}(P_1, P_2) \rrbracket_a^G(\mu) &= \llbracket P_1 \rrbracket_a^G(\mu) \ominus \bigoplus_{\substack{\mu' \sim \mu, \\ \text{dom}(\mu') \cap \text{dom}(\mu) \neq \emptyset}} \llbracket P_2 \rrbracket_a^G(\mu'), \\ \llbracket \text{PROJECT}(P_1, L) \rrbracket_a^G(\mu) &= \bigoplus_{\mu'_L = \mu} \llbracket P_1 \rrbracket_a^G(\mu'), \\ \llbracket \text{EXTEND}(P_1, v, T) \rrbracket_a^G(\mu) &= \begin{cases} \llbracket P_1 \rrbracket_a^G(\mu'), & \text{if } v \notin \text{dom}(\mu') \text{ and } \mu = \mu' \dot{\cup} \{v \mapsto \llbracket T \rrbracket^{\mu'}\}, \\ \mathbf{0}, & \text{otherwise,} \end{cases} \end{aligned}$$

where $\lfloor \cdot \rfloor: \{\top, \perp, \epsilon\} \rightarrow \{\mathbf{0}, \mathbf{1}\}$ is the function defined by taking $\lfloor \top \rfloor = \mathbf{1}$ and $\lfloor \perp \rfloor = \lfloor \epsilon \rfloor = \mathbf{0}$. Note again that this semantics is applicable only to snapshot graph patterns, which have a single temporal variable, t , and FILTER uses the snapshot semantics for filter expressions F (which is sufficient at this stage, but will be replaced with a more general semantics in Section 3.2).

We now show that the semantics induced by the semiring $\mathcal{S}_{\mathbb{T}}$ captures the snapshot-based semantics of Section 2.2. This property is called *snapshot-reducibility*, and an analogous result holds for temporal relational algebra [7]. A snapshot RDF graph $\mathcal{G}$ is called *representable* if

$$\{\tau \in \mathbb{T} \mid (s, p, o) \in \mathcal{G}(\tau)\} \in S_{\mathbb{T}}, \quad \text{for all } (s, p, o) \in \mathcal{RDF}.$$

If this is the case, then we write $\mathcal{G}_{\mathcal{S}_{\mathbb{T}}}$ for the $\mathcal{S}_{\mathbb{T}}$ -annotated RDF graph of $\mathcal{G}$:

$$\mathcal{G}_{\mathcal{S}_{\mathbb{T}}}(s, p, o) = \{\tau \in \mathbb{T} \mid (s, p, o) \in \mathcal{G}(\tau)\}, \quad \text{for all } (s, p, o) \in \mathcal{RDF}.$$

The following result says that, for representable snapshot graphs, the semantics of SPARQL induced by the semiring $\mathcal{S}_{\mathbb{T}}$ coincides with the snapshot semantics of Section 2.2.

► **Theorem 2** (snapshot-reducibility). *For a representable snapshot graph $\mathcal{G}$, snapshot graph pattern P , time point τ and solution mapping μ , we have $\tau \in \llbracket P \rrbracket_a^{\mathcal{G}_{\mathcal{S}_{\mathbb{T}}}}(\mu)$ iff $\mu \in \llbracket P \rrbracket_s^{\mathcal{G}(\tau)}$.*

Proof. The standard set semantics of SPARQL is induced by the Boolean SPM-semiring $\mathcal{B} = \langle \{\perp, \top\}, \vee, \wedge, \dot{-}, \perp, \top \rangle$, where $a \dot{-} b = a \wedge \neg b$; see, e.g., [9]. More precisely, for a snapshot RDF graph $\mathcal{G}$ and $\tau \in \mathbb{T}$, we define a $\mathcal{B}$ -annotated graph $\mathcal{G}_{\mathcal{B}}^\tau$ by taking $\mathcal{G}_{\mathcal{B}}^\tau(s, p, o) = \top$ iff $(s, p, o) \in \mathcal{G}(\tau)$. Then, by induction on the structure of P , one can show that

$$\llbracket P \rrbracket_a^{\mathcal{G}_{\mathcal{B}}^\tau}(\mu) = \top \quad \text{iff} \quad \mu \in \llbracket P \rrbracket_s^{\mathcal{G}(\tau)}. \quad (1)$$

Next, for $\tau \in \mathbb{T}$, define $h_\tau: S_{\mathbb{T}} \rightarrow \{\perp, \top\}$ by taking $h_\tau(s) = \top$ iff $\tau \in s$. Observe that this function is a homomorphism from $S_{\mathbb{T}}$ to $\mathcal{B}$, because it satisfies $h_\tau(s_1 \cup s_2) = h_\tau(s_1) \vee h_\tau(s_2)$, $h_\tau(s_1 \cap s_2) = h_\tau(s_1) \wedge h_\tau(s_2)$, $h_\tau(s_1 \setminus s_2) = h_\tau(s_1) \dot{-} h_\tau(s_2)$, $h_\tau(\emptyset) = \perp$ and $h_\tau(\mathbb{T}) = \top$. It follows that $h_\tau(\llbracket P \rrbracket_a^{\mathcal{G}_{S_{\mathbb{T}}}}(\mu)) = \llbracket P \rrbracket_a^{\mathcal{G}_{\mathcal{B}}}(\mu)$, and so $\llbracket P \rrbracket_a^{\mathcal{G}_{S_{\mathbb{T}}}}(\mu) = \top$ iff $\tau \in \llbracket P \rrbracket_a^{\mathcal{G}_{S_{\mathbb{T}}}}(\mu)$. The claim is then immediate from (1). $\blacktriangleleft$

The same temporal annotation (i.e., an element of $S_{\mathbb{T}}$) can be represented by different sets of intervals, which may be impractical (for instance, two equivalent queries may produce syntactically different results); see also [7]. This can be easily addressed by means of (temporal) *coalescing* [3]. A set of intervals $I \subseteq \mathbb{I}$ is said to be *coalesced* if each $I \in I$ is maximal, that is, there is no $I' \in \mathbb{I}$ such that $I \subset I' \subseteq \bigcup I$. Clearly, for a temporal annotation $s \in S_{\mathbb{T}}$, there is a unique finite coalesced set of intervals $C(s)$ such that $\bigcup C(s) = s$. Thus, $C(s)$ can serve as a canonical representation for s .

3 Comparing Intervals

We now extend our annotation-based semantics to a fragment of Temporal SPARQL that supports comparisons between maximal intervals, notably by means of Allen's relations. This is achieved via multiple temporal variables and existential quantification induced by the structure of EXISTS operations. For instance, in Example 1, an interval I_1 bound to the variable `?t1` should be retained if it ends before *some* interval I_2 bound to `?t2`. The comparison itself (`time:Before(?t2, ?t1)` in this example) is an atomic Boolean expression $A(t_1, \dots, t_n)$ in the grammar in Section 2.1, which we call a *temporal atom*. Its semantics can only be defined if each variable t_i is associated with a (unique) set I_i of intervals. To ensure that computing I_i remains tractable in the size of the data, we impose syntactic restrictions on Temporal SPARQL, which are described next.

3.1 Projected Temporal Variable

We introduce syntactic restrictions on the query language defined in Section 2.1: intuitively, a query must project exactly one temporal variable. More precisely, we say that a graph pattern P is *well-formed* and *binds* temporal variable t if it is a quad pattern of the form (s, p, o, t) , for some s, p and o , or

- (i) all quad patterns in P outside filter expressions bind t , and
- (ii) in each filter expression of the form $\text{EXISTS}(P')$ that occurs in P , the graph pattern P' is well-formed and binds a temporal variable *other* than t .

Restriction (ii) prevents any “implicit” temporal join induced by a graph pattern P and a subexpression $\text{EXISTS}(P')$ of P binding the same temporal variable t . This restriction is not strictly needed, but has the benefit of simplifying the semantics of the language, while not reducing its expressivity (one can always introduce an “explicit” temporal join instead). Restriction (i) on the other hand is essential: it excludes patterns that bind two or more temporal variables. As a counterexample, consider a query of the form $\text{FILTER}(\text{JOIN}(P_1, P_2), \text{EXISTS}(\text{FILTER}(P_3, A(t_1, t_3))))$, where each pattern P_i binds the temporal variable t_i . Its subpattern $\text{JOIN}(P_1, P_2)$ is not well-formed because it binds two temporal variables, t_1 and t_2 . We shall see below that coalescing a temporal annotation for such a pattern is intractable in the size of the graph (i.e., in data complexity). And apart from the coalesced form, we are not aware of any representation that specifies a unique set of intervals for t_1 (and a given solution mapping). So this rules out an efficient evaluation of $A(t_1, t_3)$.

To complete this argument, we show that coalescing is intractable even for two temporal variables. To this end, we exploit a well-known technique, already used in the context of bi-temporal databases [17], namely, a (linear-time) reduction of the following NP-hard [5] problem: given $k \in \mathbb{N}$ and a finite set $\mathbf{R} = \{R_1, \dots, R_n\}$ of rectangles, whose union is a polygon $p(\mathbf{R})$, decide whether $p(\mathbf{R})$ can be covered with k rectangles. Each rectangle $R_i \in \mathbf{R}$ can be represented as the product $I_i \times J_i$ of two intervals in the Euclidean plane. Then one can construct a graph $\mathbf{G}_{\mathbf{R}}$ that encodes each pair (I_i, J_i) as two triple annotations $\mathbf{G}_{\mathbf{R}}(\mathbf{r}_i, \mathbf{a}, \mathbf{a}) = I_i$ and $\mathbf{G}_{\mathbf{R}}(\mathbf{r}_i, \mathbf{b}, \mathbf{b}) = J_i$. Now, consider the graph pattern $P = \text{PROJECT}(\text{JOIN}((x, \mathbf{a}, \mathbf{a}, t_I), (x, \mathbf{b}, \mathbf{b}, t_J)), \emptyset)$, where $x \in \mathcal{V}$ and $t_I, t_J \in \mathcal{T}$. Observe that P is independent of $\mathbf{R}$ and k . If one is to generalise the semantics of Section 2.3 to patterns that bind two variables, then $\llbracket P \rrbracket_a^{\mathbf{G}_{\mathbf{R}}, \emptyset}$ on the solution mapping with the empty domain should give us a binary relation of the form $B = (I_1 \times J_1) \cup \dots \cup (I_n \times J_n)$; moreover, we would naturally say that a representation $\{(I_1, J_1), \dots, (I_m, J_m)\}$ of B is *coalesced* if each $I_j \times J_j$ is maximal w.r.t. set inclusion for $1 \leq j \leq m$, that is, no interval I satisfies $I_j \subset I$ and $I \times J_j \subseteq B$, and similarly for J_j . It should be clear that m is the least number of rectangles needed to cover $p(\mathbf{R})$, therefore $p(\mathbf{R})$ can be covered with k rectangles iff $m \leq k$.

Tractability can be regained by requiring that rectangles in a cover do not overlap. In this case, a minimal (w.r.t. cardinality) set of rectangles that covers $p(\mathbf{R})$ can be computed in polynomial time. Unfortunately, minimal covers are *not unique* anymore. For instance, an L-shaped polygon can be covered by two non-overlapping rectangles, but there are two alternative ways of doing so. This is why we cannot base our semantics on this weaker form of coalescing (even for two temporal variables).

3.2 Annotation-Based Semantics with Interval Comparisons

We now extend the annotation-based semantics of Section 2.3 to the remaining constructs, namely, EXISTS and temporal atoms, in such a way that it still coincides with the snapshot semantics of Section 2.2 (obviously, for snapshot graph patterns only).

In order to handle the EXISTS operator, we follow Kaminski et al. [13] and use contexts, which allows us to evaluate a pattern of the form $\text{FILTER}(P, \text{EXISTS}(P'))$ by “injecting” a solution mapping from P into P' . Formally, a *context* ξ is a partial finite-domain map from object variables $\mathcal{V}$ to the domain $\mathcal{D}$ and from temporal variables $\mathcal{T}$ to the set $\mathbb{I}$ of intervals. We adjust the semantics $\llbracket P \rrbracket_a^{\mathbf{G}}$ of graph patterns accordingly: we write $\llbracket P \rrbracket_a^{\mathbf{G}, \xi}$, where ξ is a context. We also need to extend the evaluation function to a filter expressions F to cover the cases of EXISTS and temporal atoms:

$$\begin{aligned} \llbracket \text{EXISTS}(P) \rrbracket_a^{\mathbf{G}, \xi} &= \begin{cases} \top, & \text{if } \llbracket P \rrbracket_a^{\mathbf{G}, \xi}(\mu) \neq \mathbf{0}, \text{ for some } \mu, \\ \perp, & \text{otherwise,} \end{cases} \\ \llbracket A(t_1, \dots, t_n) \rrbracket_a^{\mathbf{G}, \xi} &= \begin{cases} \epsilon, & \text{if } t_i \notin \text{dom}(\xi), \text{ for some } 1 \leq i \leq n, \\ \top, & \text{if } (\xi(t_1), \dots, \xi(t_n)) \in A, \\ \perp, & \text{otherwise;} \end{cases} \end{aligned}$$

For all other types of filter expressions, $\llbracket F \rrbracket_a^{\mathbf{G}, \xi}$ is defined analogously to $\llbracket F \rrbracket_s^{\mu}$. Note that $\llbracket F \rrbracket_a^{\mathbf{G}, \xi}$ depends not only on the context ξ (which is a solution mapping, like μ in $\llbracket F \rrbracket_s^{\mu}$), but also on the annotated RDF graph $\mathbf{G}$, as the semantics of filter expressions and graph patterns are now mutually recursive.

The full semantics of a temporal quad pattern is

$$\llbracket (s, p, o, t) \rrbracket_a^{\mathbf{G}, \xi}(\mu) = \begin{cases} \mathbf{G}((s, p, o)^{\mu \dot{\cup} \xi}), & \text{if } \text{dom}(\mu) = \text{var}((s, p, o)^{\xi}) \text{ and } (s, p, o)^{\mu \dot{\cup} \xi} \in \mathcal{RDF}, \\ \mathbf{0}, & \text{otherwise.} \end{cases}$$

All other operators but **FILTER** propagate contexts inductively. For instance, the semantics of **UNION** is given by

$$\llbracket \text{UNION}(P_1, P_2) \rrbracket_a^{G, \xi}(\mu) = \llbracket P_1 \rrbracket_a^{G, \xi}(\mu) \oplus \llbracket P_2 \rrbracket_a^{G, \xi}(\mu).$$

To support temporal atoms, the semantics of the **FILTER** operator is modified in the expected way: instead of reducing the whole annotation s for a solution mapping to the empty set, $\mathbf{0}$, we allow a **FILTER** to reduce it in a more granular way, by discarding individual intervals from the coalesced representation $C(s)$ of s (see Section 2.3). Precisely, if P binds temporal variable t , then

$$\llbracket \text{FILTER}(P, F) \rrbracket_a^{G, \xi}(\mu) = \bigoplus_{I \in C(\llbracket P \rrbracket_a^{G, \xi}(\mu))} I \otimes \llbracket F \rrbracket_a^{G, \xi \cup \mu \cup \{t \mapsto I\}}.$$

Finally, the evaluation $\llbracket P \rrbracket_a^G$ of a graph pattern P on an annotated RDF graph G is simply $\llbracket P \rrbracket_a^{G, \emptyset}$, where $\emptyset$ denotes the context with the empty domain. We illustrate these definitions with the following example.

► **Example 3.** Consider graph pattern P_1 , where

$$\begin{aligned} P_1 &= \text{FILTER}(P_2, \neg \text{EXISTS}(P_3)), \\ P_2 &= \text{JOIN}((e, \text{:worksIn}, dep_1, t_1), (dep_1, \text{:locatedIn}, \text{:barcelona}, t_1)), \\ P_3 &= \text{FILTER}(P_4, \text{time:Before}(t_2, t_1)), \\ P_4 &= \text{JOIN}((e, \text{:worksIn}, dep_2, t_2), (dep_2, \text{:locatedIn}, \text{:rome}, t_2)). \end{aligned}$$

The answers to P_1 are found in the following way. First, to obtain $\llbracket P_1 \rrbracket_a^{G, \emptyset}$, we evaluate $\llbracket P_2 \rrbracket_a^{G, \emptyset}$, which produces, say, the temporal solution mappings

$$([1999, 2003], \{e \mapsto \text{:john}, dep_1 \mapsto \text{:fin}\}), ([2000, 2003], \{e \mapsto \text{:mark}, dep_1 \mapsto \text{:hr}\});$$

note that P_2 binds temporal variable t_1 . Next, we evaluate $\llbracket \neg \text{EXISTS}(P_3) \rrbracket_a^{G, \xi_i}$ for contexts

$$\begin{aligned} \xi_1 &= \{t_1 \mapsto [1999, 2003], e \mapsto \text{:john}, dep_1 \mapsto \text{:fin}\}, \\ \xi_2 &= \{t_1 \mapsto [2000, 2003], e \mapsto \text{:mark}, dep_1 \mapsto \text{:hr}\}; \end{aligned}$$

note that these contexts map object variables e and dep_1 along with temporal variable t_1 . In order to evaluate $\llbracket P_3 \rrbracket_a^{G, \xi_i}$, we first evaluate $\llbracket P_4 \rrbracket_a^{G, \xi_i}$, which produces, say, the following sets of temporal solution mapping

$$\{([1997, 1998], \{dep_2 \mapsto \text{:hq}\})\} \text{ for } \xi_1 \quad \text{and} \quad \emptyset \text{ for } \xi_2;$$

observe that e is not in the domain of this temporal solution mapping, because its value is provided by the context, ξ_1 . Next, we can evaluate $\llbracket \text{FILTER}(P_4, \text{time:Before}(t_2, t_1)) \rrbracket_a^{G, \xi_i}$: we check $\llbracket \text{time:Before}(t_2, t_1) \rrbracket_a^{G, \xi_i}$ for the context

$$\xi'_1 = \{t_1 \mapsto [1999, 2003], e \mapsto \text{:john}, dep_1 \mapsto \text{:fin}, t_2 \mapsto [1998, 1999], dep_2 \mapsto \text{:hq}\};$$

observe that both temporal variables, t_1 and t_2 , are given values by this context – the result is positive, so $\llbracket \neg \text{EXISTS}(P_3) \rrbracket_a^{G, \xi_1}$ evaluates to $\perp$; while $\llbracket \neg \text{EXISTS}(P_3) \rrbracket_a^{G, \xi_2} = \top$.

It should be clear that this extended version of the annotation-based semantics coincides with the semantics defined in Section 2.3 for any snapshot graph pattern: such a pattern contains no **EXISTS**, therefore the context remains empty for all its sub-patterns, while the context for a filter expression becomes a solution mapping extended with a mapping for the only temporal variable, t , with the latter having no effect (since filter expressions in this fragment contain no temporal atoms, they cannot refer to t).

4 Concrete Model: Tuple-Based View

The logical model introduced in Section 2.3 and extended in Section 3.2 relies on (unions of) finite sets of intervals as annotations. While convenient to reason about the algebraic properties of query evaluation, this view does not immediately translate to an implementation over a relational database or a triple store, for two reasons:

- (i) the size of annotations (i.e., the number of intervals required to represent them) is unbounded, while a relational database or a triple store naturally manipulates tuples with a bounded or, respectively, fixed arity,
- (ii) the semantics of temporal atoms assumes coalesced annotations, but leaves effective coalescing underspecified.

This is why we follow Dignös et al. [7] and introduce a more concrete model, where each tuple carries a single interval, which addresses (i). As for (ii), below we use an explicit temporal coalescing operator TC , which maps a set of tuples to an equivalent one where intervals are coalesced. Interestingly, we shall see that TC is needed after certain SPARQL operators but is redundant for their “natural counterparts” in temporal relational algebra.

A *temporal solution mapping* (I, μ) consists of a nonempty interval $I \in \mathbb{I}$ and a solution mapping μ . For a set $\mathbf{T}$ of temporal solution mappings, we write $\overline{\mathbf{T}}$ to denote the function that maps solution mappings to their sets of intervals: $\overline{\mathbf{T}}(\mu) = \{I \mid (I, \mu) \in \mathbf{T}\}$, for every μ . The operator TC transforms a set $\mathbf{T}$ of temporal solution mappings to its coalesced form: formally, $\text{TC}(\mathbf{T})$ is the unique set of temporal solution mappings that satisfies

$$\overline{\text{TC}(\mathbf{T})}(\mu) = \mathcal{C}(\bigcup \overline{\mathbf{T}}(\mu)), \quad \text{for any } \mu;$$

in Section 5, we will discuss a possible implementation of TC in SQL.

A *temporal graph* $\mathbf{G}$ is a finite set of temporal triples $(I, (s, p, o))$, where $I \in \mathbb{I}$ is a nonempty interval and $(s, p, o) \in \mathcal{RDF}$ an RDF triple. We denote by $\overline{\mathbf{G}}$ the function that maps each triple $(s, p, o) \in \mathcal{RDF}$ to the set of intervals $\{I \mid (I, (s, p, o)) \in \mathbf{G}\}$. A temporal graph $\mathbf{G}$ is said to be *coalesced* if $\overline{\mathbf{G}}(s, p, o) = \mathcal{C}(\overline{\mathbf{G}}(s, p, o))$, for every $(s, p, o) \in \mathcal{RDF}$.

The evaluation $\llbracket P \rrbracket_i^{\mathbf{G}, \xi}$ of a graph pattern P over a coalesced temporal graph $\mathbf{G}$ with context ξ is a finite set of temporal solution mappings defined inductively in as follows:

$$\begin{aligned} \llbracket (s, p, o, t) \rrbracket_i^{\mathbf{G}, \xi} &= \{ (I, \mu) \mid \text{dom}(\mu) = \text{var}((s, p, o)^\xi), (I, (s, p, o)^{\mu \dot{\cup} \xi}) \in \mathbf{G} \}, \\ \llbracket \text{FILTER}(P_1, F) \rrbracket_i^{\mathbf{G}, \xi} &= \{ (I, \mu) \in \llbracket P_1 \rrbracket_i^{\mathbf{G}, \xi} \mid \llbracket F \rrbracket_i^{\mathbf{G}, \xi \dot{\cup} \mu \dot{\cup} \{t \mapsto I\}} = \top \}, \\ &\quad \text{where } P_1 \text{ binds temporal variable } t, \\ \llbracket \text{JOIN}(P_1, P_2) \rrbracket_i^{\mathbf{G}, \xi} &= \text{TC} \{ (I_1 \cap I_2, \mu_1 \dot{\cup} \mu_2) \mid (I_1, \mu_1) \in \llbracket P_1 \rrbracket_i^{\mathbf{G}, \xi}, (I_2, \mu_2) \in \llbracket P_2 \rrbracket_i^{\mathbf{G}, \xi}, \\ &\quad \mu_1 \sim \mu_2 \text{ and } I_1 \cap I_2 \neq \emptyset \}, \\ \llbracket \text{UNION}(P_1, P_2) \rrbracket_i^{\mathbf{G}, \xi} &= \text{TC}(\llbracket P_1 \rrbracket_i^{\mathbf{G}, \xi} \cup \llbracket P_2 \rrbracket_i^{\mathbf{G}, \xi}), \\ \llbracket \text{OPTIONAL}(P_1, P_2, F) \rrbracket_i^{\mathbf{G}, \xi} &= \text{TC}(\llbracket Q \rrbracket_i^{\mathbf{G}, \xi} \cup \{ (I', \mu) \mid (I, \mu) \in \llbracket P_1 \rrbracket_i^{\mathbf{G}, \xi}, I' \subseteq I \setminus \bigcup \mathbf{J}^\mu \}), \\ &\quad \text{where } \mathbf{J}^\mu = \{ J \mid (J, \mu \dot{\cup} \mu') \in \llbracket Q \rrbracket_i^{\mathbf{G}, \xi} \text{ for some } \mu' \sim \mu \} \\ &\quad \text{and } Q = \text{FILTER}(\text{JOIN}(P_1, P_2), F), \\ \llbracket \text{MINUS} \rrbracket_i^{\mathbf{G}, \xi}(P_1, P_2) &= \text{TC} \{ (I', \mu) \mid (I, \mu) \in \llbracket P_1 \rrbracket_i^{\mathbf{G}, \xi}, I' \subseteq I \setminus \bigcup \mathbf{J}^\mu \}, \\ &\quad \text{where } \mathbf{J}^\mu = \{ J \mid (J, \mu') \in \llbracket P_2 \rrbracket_i^{\mathbf{G}, \xi}, \mu' \sim \mu, \text{dom}(\mu') \cap \text{dom}(\mu) \neq \emptyset \}, \\ \llbracket \text{PROJECT}(P_1, L) \rrbracket_i^{\mathbf{G}, \xi} &= \text{TC} \{ (I, \mu|_L) \mid (I, \mu) \in \llbracket P_1 \rrbracket_i^{\mathbf{G}, \xi} \}, \\ \llbracket \text{EXTEND}(P_1, v, T) \rrbracket_i^{\mathbf{G}, \xi} &= \{ (I, \mu \dot{\cup} \{v \mapsto \llbracket T \rrbracket^\mu\}) \mid (I, \mu) \in \llbracket P_1 \rrbracket_i^{\mathbf{G}, \xi} \}. \end{aligned}$$

As before, the evaluation of a graph pattern P over G is $\llbracket P \rrbracket_i^{G, \emptyset}$. Again, because the definitions of graph patterns and filter expressions (with EXISTS) are mutually recursive, we also need to extend this evaluation function to a filter expression F :

$$\llbracket \text{EXISTS}(P) \rrbracket_i^{G, \xi} = \begin{cases} \top, & \text{if } \llbracket P \rrbracket_i^{G, \xi} \neq \emptyset, \\ \perp, & \text{otherwise.} \end{cases}$$

For all other types of filter expressions, $\llbracket F \rrbracket_i^{G, \xi}$ is defined analogously to $\llbracket F \rrbracket_a^{G, \xi}$.

We first observe that, similarly to temporal relation algebra, some operations, namely UNION and PROJECT, naturally require an additional TC operator in their semantics [3, Theorem 4.1]; on the other hand, in line with temporal relation algebra, FILTER does not require TC [3, Theorem 4.2]. Rather interestingly, the compatibility condition for JOIN in SPARQL leads to a more subtle definition of its semantics, which surprisingly also requires an additional TC operator (which is in stark contrast to its temporal relational algebra counterpart, INNER JOIN, which preserves coalescing without any additional measures [3, Theorem 4.2]). We illustrate the issue on the following example.

► **Example 4.** Consider $\llbracket \text{JOIN}(P_1, P_2) \rrbracket_i^{G, \emptyset}$ with

$$\llbracket P_1 \rrbracket_i^{G, \emptyset} = \left\{ \begin{array}{l} ([0, 1), \{x \mapsto :a\}) \\ ([1, 2), \{y \mapsto :b\}) \end{array} \right\}, \quad \llbracket P_2 \rrbracket_i^{G, \emptyset} = \{([0, 2), \{x \mapsto :a, y \mapsto :b\})\}.$$

Both solution mappings in $\llbracket P_1 \rrbracket_i^{G, \emptyset}$ are compatible with the solution mapping in $\llbracket P_2 \rrbracket_i^{G, \emptyset}$, and their intervals overlap. So the simple expression $(I_1 \cap I_2, \mu_1 \dot{\cup} \mu_2)$ would give us the following two temporal solution mappings:

$$([0, 1), \{x \mapsto :a, y \mapsto :b\}) \quad \text{and} \quad ([1, 2), \{x \mapsto :a, y \mapsto :b\}),$$

which are evidently not coalesced.

The semantics of both OPTIONAL and MINUS obviously requires the TC operator as the $I' \subseteq I \setminus \bigcup J^\mu$ component gives us all possible I' (not coalesced) and is in fact even potentially unbounded. While we could push the TC operator inside the set formation for MINUS:

$$\llbracket \text{MINUS} \rrbracket_i^{G, \xi}(P_1, P_2) = \{(I', \mu) \mid (I, \mu) \in \llbracket P_1 \rrbracket_i^{G, \xi}, I' \in \mathcal{C}(I \setminus \bigcup J^\mu)\},$$

the TC operator in OPTIONAL would still be required due to $\cup$, as our next example shows.

► **Example 5.** Consider $\llbracket \text{OPTIONAL}(P_1, P_2, F) \rrbracket_i^{G, \emptyset}$ with $F = \neg \text{BOUND}(z)$ and

$$\llbracket P_1 \rrbracket_i^{G, \emptyset} = \left\{ \begin{array}{l} ([0, 1), \{x \mapsto :a\}) \\ ([1, 2), \{x \mapsto :a, y \mapsto :b\}) \end{array} \right\}, \quad \llbracket P_2 \rrbracket_i^{G, \emptyset} = \left\{ \begin{array}{l} ([0, 1), \{x \mapsto :a, y \mapsto :b\}) \\ ([1, 2), \{x \mapsto :a, z \mapsto :c\}) \end{array} \right\}.$$

Evaluating $\llbracket Q \rrbracket_i^{G, \emptyset}$, for $Q = \text{FILTER}(\text{JOIN}(P_1, P_2), F)$, gives us

$$([0, 1), \{x \mapsto :a, y \mapsto :b\}).$$

On the other hand, since $([1, 2), \mu) \in \llbracket P_1 \rrbracket_i^{G, \emptyset}$ but $J^\mu = \{[0, 1)\}$, for $\mu = \{x \mapsto :a, y \mapsto :b\}$, the second component of the union produces

$$([1, 2), \{x \mapsto :a, y \mapsto :b\}),$$

which means that the set with the two temporal solution mappings above needs be coalesced.

■ **Listing 2** Married Athletes query in Temporal SPARQL (for prefix declarations, see Listing 1).

```
SELECT DISTINCT ?x ?z ?c ?t
WHERE {
  << ?x :playsFor ?c >> time:hasTime ?t .
  << ?x :marriedTo ?z >> time:hasTime ?t .
}
```

Note, however, that, in some concrete graph patterns, the TC operator could be redundant. For example, if both P_1 and P_2 in $\text{JOIN}(P_1, P_2)$ are temporal basic graph patterns (BGPs), that is, JOINS of temporal quad patterns, then the TC operator in $\llbracket \text{JOIN}(P_1, P_2) \rrbracket_i^{\mathcal{G}, \xi}$ is not needed (this can be shown by using the same argument as for the INNER JOIN in temporal relational algebra [3, Theorem 4.2]). The same applies to combinations of JOIN and FILTER patterns. A more in-depth study of sufficient conditions for optimisations eliminating superfluous TC operators is left for future work.

We now show that the defined semantics is correct, meaning that it coincides with the one defined for the logical model in Section 3.2. Let $\mathcal{G}_{S_T}$ denote the S_T -annotated graph associated with $\mathcal{G}$, defined by taking $\mathcal{G}_{S_T}(s, p, o) = \bigcup \overline{\mathcal{G}}(s, p, o)$, for all $(s, p, o) \in \mathcal{RDF}$. Then we have the following result (shown in Appendix A).

► **Theorem 6.** *For any coalesced temporal graph $\mathcal{G}$, graph pattern P , context ξ and solution mapping μ , we have $\llbracket P \rrbracket_i^{\mathcal{G}, \xi}(\mu) = \mathcal{C}(\llbracket P \rrbracket_a^{\mathcal{G}_{S_T}, \xi}(\mu))$.*

5 Experiments

For our experiments, we take the setting of *virtual* knowledge graphs, where the data source is relational and exposed as an RDF graph via a mapping. Input queries in Temporal SPARQL are rewritten by the system (using the mapping) into equivalent SQL queries to be evaluated over the source, thus leveraging the existing capabilities of the relational world. In addition to its flexibility, this approach is arguably better suited to joins over temporal annotations than a triple store, as we shall see below. Besides, as shown by Zhou et al. [19], coalescing can be efficiently performed in SQL by means of window functions. So, we present here two scenarios, complete with data and input temporal queries, and show in each case how such a system could produce equivalent SQL queries and what evaluation time can be achieved, to highlight the promise of our approach.

The experiments were run on a Lenovo ThinkPad X13 Gen 5 with 32GB RAM of LPDDR5 memory running at 7467 MT/s, an Intel Core Ultra 7 155U CPU, under Ubuntu 25.10, using Linux kernel version 6.11.0-17. The OS and data are running on an SSD. The datasets were loaded on a local Postgres v. 17.9 installation. The Temporal SPARQL queries were translated by hand; the resulting SQL queries are given in this section as well as in Appendix B. We report the evaluation times in Table 1.

5.1 Scenario 1: Married Athletes on Wikidata

Chekol et al. [4] develop bespoke algorithms for computing temporal joins in SPARQL because one of their motivating examples demonstrates a clear need for a better temporal join implementation: existing SPARQL endpoints show poor performance. The query uses Wikidata and asks for all people and their spouses (predicate `wdt:marriedTo/P56`) such that they were married at some point while they represented a team/club (`wdt:playsFor/P54`). In fact, to

■ **Table 1** Evaluation times of Temporal SPARQL queries translated to SQL.

query	evaluation time (ms)
Married Athletes Query (translated)	623.955
Flights Query 1 (translated)	1 054.958
Flights Query 2 (translated)	204 458.840

■ **Table 2** WikiData cardinality info for the join selectivity in the Married Athletes query.

basic graph pattern	cardinality
<code>?x p:playsFor ?r1.</code>	1 762 340
<code>?x p:marriedTo ?r2.</code>	920 713
<code>?x p:playsFor ?r1. ?x p:marriedTo ?r2 .</code>	12 593
<code>?r1 pq:startTime ?s1 .</code>	10 641 734
<code>?r1 pq:endTime ?e1 .</code>	6 457 762
<code>?r1 pq:startTime ?s1 . ?r1 pq:endTime ?e1 .</code>	timeout

refer to temporal annotations (`pq:startTime/P580` and `pq:endTime/P582`) of the statements, one has to access *reified* statements using pairs of predicates `p:marriedTo/ps:marriedTo` and `p:playsFor/ps:playsFor`, so the following basic graph pattern is used:

```
?x p:playsFor ?r1. ?r1 ps:playsFor ?z; pq:startTime ?s1; pq:endTime ?e1.
?x p:marriedTo ?r2. ?r2 ps:marriedTo ?y; pq:startTime ?s2; pq:endTime ?e2.
```

where `;` separates predicate-object pairs that share the same subject. The authors reported that this query timed out over the public Wikidata endpoint. We repeated their experiment on the present-day public Wikidata endpoint: it actually does evaluate, but takes 45.519s to run⁵. Of course, this does not contradict the findings, as the underlying systems and software are likely to have changed, but nevertheless nearly a minute for such a simple query is not very impressive. To understand the challenge, we present some statistics in Table 2. The join size is really dominated by the initial join between relations `p:playsFor` and `p:marriedTo`. Of the $1\,762\,340 \times 920\,713 \approx 1.5 \cdot 10^{12}$ possible combinations, only minuscule 12 593 end up in the join result. In addition, the join over the start (`pq:startTime`) and end (`pq:endTime`) points, i.e., the objects that have both some start and end point, already times out on its own. Looking at the cardinality of the two component triple patterns, we know that the join result has at most $\min\{10\,641\,734, 6\,457\,762\} = 6\,457\,762$ possible pairs, since any given object can only have a single start and a single end point in time.

We can see in Table 1 our SQL version of this query (shown in Listing 3) only needs 624ms to evaluate in Postgres, and is thus multiple orders of magnitude faster than the SPARQL query with reification. For this experiment, we extracted the information on athletes and married couples from Wikidata into two SQL tables, `athletes` and `married`, with 1 212 855 and 404 486 rows, respectively. Note that about 31% of the `wdt:playsFor` relationships have no `pq:startTime` annotation, and so are discarded; on the other hand, we use the current date for the end of an interval if there is a `pq:startTime` but no `pq:endTime`. The same applies to `wdt:marriedTo`, where, in addition, since it is a symmetric property, we use two columns, `spousea` or `spouseb`, in the `married` table to halve the number of rows.

⁵ <https://w.wiki/NBcE>

■ **Listing 3** SQL translation of the Married Athletes query in Listing 2.

```
SELECT a.person, m.spouseb AS spouse, a.club,
       GREATEST(a.s, m.s) AS ts, LEAST(a.e, m.e) AS te
FROM   athletes AS a, married AS m
WHERE  a.person = m.spousea AND GREATEST(a.s, m.s) < LEAST(a.e, m.e)
UNION ALL
SELECT a.person, m.spousea AS spouse, a.club,
       GREATEST(a.s, m.s) AS ts, LEAST(a.e, m.e) AS te
FROM   athletes AS a, married AS m
WHERE  a.person = m.spouseb AND GREATEST(a.s, m.s) < LEAST(a.e, m.e)
```

The query in Listing 3 is obtained by a rather straightforward translation of the Temporal SPARQL query in Listing 2. We encode the temporal join as an inner join over the **person** and **spouse** columns with an additional temporal overlap condition: the filter condition `GREATEST(a.s, m.s) < LEAST(a.e, m.e)` ensures that intervals $[a.s, a.e)$ and $[m.s, m.e)$ overlap, while `GREATEST(a.s, m.s)` and `LEAST(a.e, m.e)` produce the endpoints for the intersection of these intervals; note that SQL functions `GREATEST` and `LEAST` return the maximum and the minimum in a given list of values. Finally, since `wdt:marriedTo` is a symmetric property, the results of two subqueries that obtain spouses from **spousea** and **spouseb** columns are combined with `UNION ALL`.

The SQL version can be so much faster since it can limit itself to a simple two-way join over the two tables of interest, whereas the SPARQL query essentially has to consider a BGP of 8 triple patterns, each of which involves millions triples completely unrelated to the (very small) query output (2 353 matches).

5.2 Scenario 2: Flight Dataset

The second scenario is over an excerpt of public international flight data for November 2014 [1], which has previously been used as a dataset for experiments on interval join algorithms [16]. The dataset consists of 57 585 entries.

Over this flights dataset, we are considering two queries. Listing 4 shows a Temporal SPARQL query that considers all combinations of pairs of flights to **FRA** (Frankfurt Airport) and **MUC** (Munich Airport), which are in the air at the same time, meaning that their intervals between flight departures and arrivals intersect. We refer to these interval as the “flight intervals”. Listing 5 extends the first query by an `OPTIONAL` that adds a third flight to **CDG** (Paris Charles de Gaulle International Airport). In each query, the time interval output is the flight interval that the two (three, respectively) flights have in common.

In Table 1, we can see that the SQL translation of the query in Listing 4 runs particularly fast, needing a mere second. This is no surprise, since, similarly to Listing 3, the temporal join can be translated into a simple self-join over the **flights** table with an additional inequality. In comparison, the SQL translation of the query in Listing 5 involves a temporal `LEFT JOIN` and is much slower. This too comes at no surprise, when one considers the size of the translated query. The encoding of temporal `LEFT JOIN` follows the example translations given by Dignös et al. [6] and requires a number of common table expressions (CTEs) with SQL window functions.

■ **Listing 4** Flights Query 1 (for prefix declarations, see Listing 1).

```
SELECT DISTINCT ?x ?t
WHERE {
  << ?x :fap 'FRA' >> time:hasTime ?t .
  << ?z :fap 'MUC' >> time:hasTime ?t .
}
```

■ **Listing 5** Flights Query 2 (for prefix declarations, see Listing 1).

```
SELECT DISTINCT ?x ?w ?t
WHERE {
  << ?x :fap 'FRA' >>      time:hasTime ?t .
  << ?z :fap 'MUC' >>      time:hasTime ?t .
  OPTIONAL {
    << ?w :fap 'CDG' >> time:hasTime ?t .
  }
}
```

6 Conclusion

We equipped SPARQL with temporal abilities, for a fragment of the language that we call Temporal SPARQL, with suitable restrictions to support interval comparisons (using, e.g., Allen’s relations) while retaining tractable query evaluation. We provided a formal semantics for this language that complies with the specifics of the SPARQL algebra, which is based on the compatibility of partial solution mapping. Analogously to what was done by Dignös et al. [7] for relational algebra, this semantics comes in three flavours, with different levels of abstraction, where time is presented as time points, unions of intervals and single intervals, respectively. The third model operates directly on interval-based temporal data. We concluded the paper with a feasibility study, showing that Temporal SPARQL can be translated to SQL, and that this solution may outperform triple stores.

A natural continuation of this work is extending OBDA systems with the ability to express queries over interval-based temporal data. Our paper has identified Temporal SPARQL as a strong candidate input language of such a system, and the topic is of high interest to the OBDA community, albeit highly non-trivial. As for the language itself, even though this fragment can express a number of interesting temporal queries, it does not handle *aggregation* over time, for instance, getting the maximal measurement within a certain time span, or counting events within a certain period. A careful investigation is needed to understand whether this can be achieved without compromising the desirable properties of this fragment.

References

- 1 Andreas Behrend and Gereon Schüller. A case study in optimizing continuous queries using the magic update technique. In *Proc. of the 26th Int. Conf. on Scientific and Statistical Database Management, SSDBM’14*, pages 31:1–31:4. ACM, 2014. doi:10.1145/2618243.2618285.
- 2 Michael H. Böhlen, Anton Dignös, Johann Gamper, and Christian S. Jensen. Temporal data management – an overview. In *Tutorial Lectures of the 7th European Summer School on Business Intelligence and Big Data (eBISS)*, volume 324 of *Lecture Notes in Business Information Processing*, pages 51–83. Springer-Verlag, 2017. doi:10.1007/978-3-319-96655-7_3.

- 3 Michael H. Böhlen, Richard T. Snodgrass, and Michael D. Soo. Coalescing in temporal databases. In *Proc. of 22nd Int. Conf. on Very Large Data Bases, VLDB'96*, pages 180–191. Morgan Kaufmann, 1996. URL: <http://www.vldb.org/conf/1996/P180.PDF>.
- 4 Melisachew Wudage Chekol, Giuseppe Pirrò, and Heiner Stuckenschmidt. Fast interval joins for temporal SPARQL queries. In *Companion of the 2019 World Wide Web Conference, WWW 2019*, pages 1148–1154. ACM, 2019. doi:10.1145/3308560.3314997.
- 5 Joseph C. Culberson and Robert A. Reckhow. Covering polygons is hard (preliminary abstract). In *Proc. of the 29th Annual Symposium on Foundations of Computer Science, SFCS'88*, pages 601–611. IEEE Computer Society, 1988. doi:10.1109/SFCS.1988.21976.
- 6 Anton Dignös, Boris Glavic, Xing Niu, Michael H. Böhlen, and Johann Gamper. Snapshot semantics for temporal multiset relations (extended version). *CoRR*, abs/1902.04938, 2019. doi:10.48550/arXiv.1902.04938.
- 7 Anton Dignös, Boris Glavic, Xing Niu, Johann Gamper, and Michael H. Böhlen. Snapshot semantics for temporal multiset relations. *Proc. VLDB Endow.*, 12(6):639–652, 2019. doi:10.14778/3311880.3311882.
- 8 Floris Geerts and Antonella Poggi. On database query languages for k-relations. *Journal of Applied Logic*, 8(2):173–185, 2010. doi:10.1016/j.jal.2009.09.001.
- 9 Floris Geerts, Thomas Unger, Grigoris Karvounarakis, Irini Fundulaki, and Vassilis Christophides. Algebraic structures for capturing the provenance of SPARQL queries. *J. ACM*, 63(1):7:1–7:63, 2016. doi:10.1145/2810037.
- 10 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proc. of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'07*, pages 31–40. ACM, 2007. doi:10.1145/1265530.1265535.
- 11 Olaf Hartig and Bryan Thompson. Foundations of an alternative approach to reification in RDF. *CoRR*, abs/1406.3399, 2014. doi:10.48550/arXiv.1406.3399.
- 12 Elem Güzel Kalayci, Sebastian Brandt, Diego Calvanese, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyashev. Ontology-based access to temporal data with ontop: A framework proposal. *Int. J. Appl. Math. Comput. Sci.*, 29(1):17–30, 2019. doi:10.2478/AMCS-2019-0002.
- 13 Mark Kaminski, Egor V. Kostylev, and Bernardo Cuenca Grau. Query nesting, assignment, and aggregation in SPARQL 1.1. *ACM Trans. Database Syst.*, 42(3):17:1–17:46, 2017. doi:10.1145/3083898.
- 14 Szymon Klarman and Thomas Meyer. Querying temporal databases via OWL 2 QL. In *Proc. of the 8th Int. Conf. on Web Reasoning and Rule Systems, RR 2014*, Lecture Notes in Computer Science, pages 92–107. Springer, 2014. doi:10.1007/978-3-319-11113-1_7.
- 15 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009. doi:10.1145/1567274.1567278.
- 16 Danila Piatov, Sven Helmer, and Anton Dignös. An interval join optimized for modern hardware. In *Proc. of the 32nd IEEE Int. Conf. on Data Engineering, ICDE 2016*, pages 1098–1109. IEEE Computer Society, 2016. doi:10.1109/ICDE.2016.7498316.
- 17 David Toman. Point-based temporal extensions of SQL and their efficient implementation. In *Temporal Databases: Research and Practice*, volume 1399 of *Lecture Notes in Computer Science*, pages 211–237. Springer, 1998. doi:10.1007/BFb0053704.
- 18 Guohui Xiao, Roman Kontchakov, Benjamin Cogrel, Diego Calvanese, and Elena Botoeva. Efficient handling of SPARQL OPTIONAL for OBDA. In *Proc. of the 17th Int. Conf. on The Semantic Web, ISWC 2018, Part I*, volume 11136 of *Lecture Notes in Computer Science*, pages 354–373. Springer, 2018. doi:10.1007/978-3-030-00671-6_21.
- 19 Gang Zhou, Yafeng Wu, Ting Yan, Tian He, Chengdu Huang, John A. Stankovic, and Tarek F. Abdelzaher. A multifrequency mac specially designed for wireless sensor network applications. *ACM Trans. Embed. Comput. Syst.*, 9(4):39:1–39:41, 2010. doi:10.1145/1721695.1721705.
- 20 Xin Zhou, Fusheng Wang, and Carlo Zaniolo. Efficient temporal coalescing query support in relational database systems. In *Proc. of the 17th Int. Conf. on Database and Expert Systems Applications, DEXA 2006*, volume 4080 of *Lecture Notes in Computer Science*, pages 676–686. Springer, 2006. doi:10.1007/11827405_66.

A Proof of Theorem 6

We show that for a coalesced temporal graph G , context ξ and solution mapping μ ,

$$\begin{aligned}\overline{\llbracket P \rrbracket_i^{G,\xi}(\mu)} &= C(\llbracket P \rrbracket_a^{G_{S_T},\xi}(\mu)), \text{ for any graph pattern } P, \\ \llbracket F \rrbracket_i^{G,\mu} &= \llbracket F \rrbracket_a^{G_{S_T},\mu}, \text{ for any filter expression } F,\end{aligned}$$

by mutual induction on the structure of P and F . To avoid notational clutter, we fix G and omit G and G_{S_T} as superscripts, i.e., we write $\llbracket P \rrbracket_i^\xi$, $\llbracket P \rrbracket_a^\xi$, $\llbracket F \rrbracket_i^\mu$ and $\llbracket F \rrbracket_a^\mu$.

■ $P = (s, p, o, t)$.

- If $\text{dom}(\mu) \neq \text{var}((s, p, o)^\xi)$ or $(s, p, o)^{\mu \dot{\cup} \xi} \notin \mathcal{RDF}$, then $\overline{\llbracket P \rrbracket_i^\xi(\mu)} = \emptyset$ and $\llbracket P \rrbracket_a^\xi(\mu) = \emptyset$.
- Otherwise, we have

$$\begin{aligned}C(\llbracket (s, p, o) \rrbracket_a^\xi(\mu)) &= C(G_{S_T}((s, p, o)^{\mu \dot{\cup} \xi})) && (\llbracket \cdot \rrbracket_a^\xi \text{ def.}) \\ &= C(\bigcup \overline{G}((s, p, o)^{\mu \dot{\cup} \xi})) && (G_{S_T} \text{ def.}) \\ &= \bigcup \overline{G}((s, p, o)^{\mu \dot{\cup} \xi}) && (G \text{ is coalesced}) \\ &= \overline{\llbracket (s, p, o) \rrbracket_i^\xi(\mu)}. && (\llbracket \cdot \rrbracket_i^\xi \text{ def.})\end{aligned}$$

■ $P = \text{FILTER}(P_1, F)$, where P_1 binds temporal variable t .

$$\begin{aligned}C(\llbracket \text{FILTER}(P_1, F) \rrbracket_a^\xi(\mu)) &= C(\bigcup \{I \in C(\llbracket P_1 \rrbracket_a^\xi(\mu)) \mid \llbracket F \rrbracket_a^{\xi \dot{\cup} \mu \dot{\cup} \{t \mapsto I\}} = \top\}) && (\llbracket \cdot \rrbracket_a^\xi \text{ def.}) \\ &= \{I \in C(\llbracket P_1 \rrbracket_a^\xi(\mu)) \mid \llbracket F \rrbracket_a^{\xi \dot{\cup} \mu \dot{\cup} \{t \mapsto I\}} = \top\} && (\dagger) \\ &= \{I \in \overline{\llbracket P_1 \rrbracket_i^\xi(\mu)} \mid \llbracket F \rrbracket_i^{\xi \dot{\cup} \mu \dot{\cup} \{t \mapsto I\}} = \top\} && (\text{IH}) \\ &= \overline{\{(I, \mu') \in \llbracket P_1 \rrbracket_i^\xi \mid \llbracket F \rrbracket_i^{\xi \dot{\cup} \mu' \dot{\cup} \{t \mapsto I\}} = \top\}}(\mu) && (\dot{\cup} \text{ def.}) \\ &= \overline{\llbracket \text{FILTER}(P_1, F) \rrbracket_i^\xi(\mu)}, && (\llbracket \cdot \rrbracket_i^\xi \text{ def.})\end{aligned}$$

where $(\dagger)$ holds because a subset of a coalesced set of intervals is also coalesced.

■ $P = \text{UNION}(P_1, P_2)$.

$$\begin{aligned}C(\llbracket \text{UNION}(P_1, P_2) \rrbracket_a^\xi(\mu)) &= C(\llbracket P_1 \rrbracket_a^\xi(\mu) \cup \llbracket P_2 \rrbracket_a^\xi(\mu)) && (\llbracket \cdot \rrbracket_a^\xi \text{ def.}) \\ &= C(\bigcup C(\llbracket P_1 \rrbracket_a^\xi(\mu)) \cup \bigcup C(\llbracket P_2 \rrbracket_a^\xi(\mu))) && (s = \bigcup C(s)) \\ &= C(\bigcup \overline{\llbracket P_1 \rrbracket_i^\xi(\mu)} \cup \bigcup \overline{\llbracket P_2 \rrbracket_i^\xi(\mu)}) && (\text{IH}) \\ &= C(\bigcup (\overline{\llbracket P_1 \rrbracket_i^\xi(\mu)} \cup \overline{\llbracket P_2 \rrbracket_i^\xi(\mu)})) && (\dot{\cup} \text{ def.}) \\ &= \overline{\text{TC}(\llbracket P_1 \rrbracket_i^\xi \cup \llbracket P_2 \rrbracket_i^\xi)(\mu)} && (\text{TC def.}) \\ &= \overline{\llbracket \text{UNION}(P_1, P_2) \rrbracket_i^\xi(\mu)}. && (\llbracket \cdot \rrbracket_i^\xi \text{ def.})\end{aligned}$$

■ $P = \text{JOIN}(P_1, P_2)$.

$$\begin{aligned}C(\llbracket \text{JOIN}(P_1, P_2) \rrbracket_a^\xi(\mu)) &= C\left(\bigcup_{\mu_1 \dot{\cup} \mu_2 = \mu} [\llbracket P_1 \rrbracket_a^\xi(\mu_1) \cap \llbracket P_2 \rrbracket_a^\xi(\mu_2)]\right) && (\llbracket \cdot \rrbracket_a^\xi \text{ def.}) \\ &= C\left(\bigcup_{\mu_1 \dot{\cup} \mu_2 = \mu} \left[\left(\bigcup C(\llbracket P_1 \rrbracket_a^\xi(\mu_1))\right) \cap \left(\bigcup C(\llbracket P_2 \rrbracket_a^\xi(\mu_2))\right)\right]\right) && (s = \bigcup C(s))\end{aligned}$$

$$\begin{aligned}
&= \mathsf{C} \left(\bigcup_{\mu_1 \dot{\cup} \mu_2 = \mu} \left[\left(\bigcup \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu_1) \right) \cap \left(\bigcup \overline{\llbracket P_2 \rrbracket_i^\xi}(\mu_2) \right) \right] \right) \quad (\text{IH}) \\
&= \mathsf{C} \left(\bigcup_{\mu_1 \dot{\cup} \mu_2 = \mu} \left[\left(\bigcup \{ I_1 \mid (I_1, \mu_1) \in \llbracket P_1 \rrbracket_i^\xi \} \right) \cap \left(\bigcup \{ I_2 \mid (I_2, \mu_2) \in \llbracket P_2 \rrbracket_i^\xi \} \right) \right] \right) \quad (\cdot \text{ def.}) \\
&= \mathsf{C} \left(\bigcup_{\mu_1 \dot{\cup} \mu_2 = \mu} \bigcup \{ I_1 \cap I_2 \mid (I_1, \mu_1) \in \llbracket P_1 \rrbracket_i^\xi, (I_2, \mu_2) \in \llbracket P_2 \rrbracket_i^\xi \} \right) \quad (\cap \text{ distr. over } \cup) \\
&= \mathsf{C} \left(\bigcup \{ I_1 \cap I_2 \mid \mu_1 \dot{\cup} \mu_2 = \mu, (I_1, \mu_1) \in \llbracket P_1 \rrbracket_i^\xi, (I_2, \mu_2) \in \llbracket P_2 \rrbracket_i^\xi \} \right) \quad (\cup \text{ assoc.}) \\
&= \mathsf{C} \left(\bigcup \{ (I_1 \cap I_2, \mu_1 \dot{\cup} \mu_2) \mid \llbracket P_1 \rrbracket_i^\xi \ni (I_1, \mu_1) \sim (I_2, \mu_2) \in \llbracket P_2 \rrbracket_i^\xi \}(\mu) \right) \quad (\cdot \text{ def.}) \\
&= \overline{\text{TC} \{ (I_1 \cap I_2, \mu_1 \dot{\cup} \mu_2) \mid \llbracket P_1 \rrbracket_i^\xi \ni (I_1, \mu_1) \sim (I_2, \mu_2) \in \llbracket P_2 \rrbracket_i^\xi \}(\mu)} \quad (\text{TC def.}) \\
&= \overline{\llbracket \text{JOIN}(P_1, P_2) \rrbracket_i^\xi}(\mu), \quad (\llbracket \cdot \rrbracket_i^\xi \text{ def.})
\end{aligned}$$

where we write $(I_1, \mu_1) \sim (I_2, \mu_2)$ if $\mu_1 \sim \mu_2$ and $I_1 \cap I_2 \neq \emptyset$.

■ $P = \text{MINUS}(P_1, P_2)$.

$$\begin{aligned}
&\mathsf{C} \left(\llbracket \text{MINUS}(P_1, P_2) \rrbracket_a^\xi(\mu) \right) \\
&= \mathsf{C} \left(\llbracket P_1 \rrbracket_a^\xi(\mu) \setminus \bigcup_{\substack{\mu_2 \sim \mu \\ \text{dom}(\mu_2) \cap \text{dom}(\mu) \neq \emptyset}} \llbracket P_2 \rrbracket_a^\xi(\mu_2) \right) \quad (\llbracket \cdot \rrbracket_a^\xi \text{ def.}) \\
&= \mathsf{C} \left(\bigcup \mathsf{C} \left(\llbracket P_1 \rrbracket_a^\xi(\mu) \right) \setminus \bigcup_{\substack{\mu_2 \sim \mu \\ \text{dom}(\mu_2) \cap \text{dom}(\mu) \neq \emptyset}} \bigcup \mathsf{C} \left(\llbracket P_2 \rrbracket_a^\xi(\mu_2) \right) \right) \quad (s = \bigcup \mathsf{C}(s)) \\
&= \mathsf{C} \left(\bigcup \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu) \setminus \bigcup_{\substack{\mu_2 \sim \mu \\ \text{dom}(\mu_2) \cap \text{dom}(\mu) \neq \emptyset}} \bigcup \overline{\llbracket P_2 \rrbracket_i^\xi}(\mu_2) \right) \quad (\text{IH}) \\
&= \mathsf{C} \left(\bigcup \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu) \setminus \bigcup_{\substack{\mu_2 \sim \mu \\ \text{dom}(\mu_2) \cap \text{dom}(\mu) \neq \emptyset}} \bigcup \{ J \mid (J, \mu_2) \in \llbracket P_2 \rrbracket_i^\xi \} \right) \quad (\cdot \text{ def.}) \\
&= \mathsf{C} \left(\bigcup \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu) \setminus \bigcup \mathbf{J}^\mu \right) \quad (\mathbf{J}^\mu \text{ def.}) \\
&= \mathsf{C} \left(\bigcup \{ I_1 \setminus \bigcup \mathbf{J}^\mu \mid I_1 \in \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu) \} \right) \quad (\setminus \text{ right-distr. over } \cup) \\
&= \mathsf{C} \left(\bigcup \{ I \in \mathbb{I} \mid I_1 \in \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu), I \subseteq I_1 \setminus \bigcup \mathbf{J}^\mu \} \right) \quad (\dagger) \\
&= \mathsf{C} \left(\bigcup \{ I \in \mathbb{I} \mid (I_1, \mu_1) \in \llbracket P_1 \rrbracket_i^\xi, I \subseteq I_1 \setminus \bigcup \mathbf{J}^{\mu_1} \}(\mu) \right) \quad (\cdot \text{ def.}) \\
&= \overline{\text{TC} \{ (I, \mu_1) \mid (I_1, \mu_1) \in \llbracket P_1 \rrbracket_i^\xi, I \subseteq I_1 \setminus \bigcup \mathbf{J}^{\mu_1} \}(\mu)} \quad (\text{TC def.}) \\
&= \overline{\llbracket \text{MINUS}(P_1, P_2) \rrbracket_i^\xi}(\mu). \quad (\llbracket \cdot \rrbracket_i^\xi \text{ def.})
\end{aligned}$$

where $(\dagger)$ holds because $I' = \bigcup (2^{I'} \cap \mathbb{I})$, for any interval I' (in particular for $I' = I_1 \setminus \bigcup \mathbf{J}^\mu$).

■ $P = \text{OPTIONAL}(P_1, P_2, F)$ with $Q = \text{FILTER}(\text{JOIN}(P_1, P_2), F)$.

$$\begin{aligned}
&\mathsf{C} \left(\llbracket \text{OPTIONAL}(P_1, P_2, F) \rrbracket_a^\xi(\mu) \right) \\
&= \mathsf{C} \left(\llbracket Q \rrbracket_a^\xi(\mu) \cup \left(\llbracket P_1 \rrbracket_a^\xi(\mu) \setminus \bigcup_{\mu_2 \sim \mu} \llbracket Q \rrbracket_a^\xi(\mu \dot{\cup} \mu_2) \right) \right) \quad (\llbracket \cdot \rrbracket_a^\xi \text{ def.}) \\
&= \mathsf{C} \left(\bigcup \mathsf{C} \left(\llbracket Q \rrbracket_a^\xi(\mu) \right) \cup \left(\bigcup \mathsf{C} \left(\llbracket P_1 \rrbracket_a^\xi(\mu) \right) \setminus \bigcup_{\mu_2 \sim \mu} \bigcup \mathsf{C} \left(\llbracket Q \rrbracket_a^\xi(\mu \dot{\cup} \mu_2) \right) \right) \right) \quad (s = \bigcup \mathsf{C}(s)) \\
&= \mathsf{C} \left(\bigcup \overline{\llbracket Q \rrbracket_i^\xi}(\mu) \cup \left(\bigcup \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu) \setminus \bigcup_{\mu_2 \sim \mu} \bigcup \overline{\llbracket Q \rrbracket_i^\xi}(\mu \dot{\cup} \mu_2) \right) \right) \quad (\text{IH})
\end{aligned}$$

$$\begin{aligned}
&= \mathsf{C} \left(\bigcup \overline{\llbracket Q \rrbracket_i^\xi}(\mu) \cup \left(\bigcup \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu) \setminus \bigcup_{\mu_2 \sim \mu} \bigcup \{J \mid (J, \mu \dot{\cup} \mu_2) \in \llbracket Q \rrbracket_i^\xi\} \right) \right) & (\bar{\cdot} \text{ def.}) \\
&= \mathsf{C} \left(\bigcup \overline{\llbracket Q \rrbracket_i^\xi}(\mu) \cup \left(\bigcup \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu) \setminus \bigcup \mathbf{J}^\mu \right) \right) & (\mathbf{J}^\mu \text{ def.}) \\
&= \mathsf{C} \left(\bigcup \left(\overline{\llbracket Q \rrbracket_i^\xi}(\mu) \cup \{I_1 \setminus \bigcup \mathbf{J}^\mu \mid I_1 \in \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu)\} \right) \right) & (\setminus \text{ right-distr. over } \cup) \\
&= \mathsf{C} \left(\bigcup \left(\overline{\llbracket Q \rrbracket_i^\xi}(\mu) \cup \{I \in \mathbb{I} \mid I_1 \in \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu), I \subseteq I_1 \setminus \bigcup \mathbf{J}^\mu\} \right) \right) & (\dagger) \\
&= \mathsf{C} \left(\bigcup \left(\overline{\llbracket Q \rrbracket_i^\xi} \cup \{(I, \mu_1) \mid (I, \mu_1) \in \llbracket P_1 \rrbracket_i^\xi, I \subseteq I_1 \setminus \bigcup \mathbf{J}^{\mu_1}\} \right) \right) & (\bar{\cdot} \text{ def.}) \\
&= \overline{\mathsf{TC} \left(\llbracket Q \rrbracket_i^\xi \cup \{(I, \mu_1) \mid (I, \mu_1) \in \llbracket P_1 \rrbracket_i^\xi, I \subseteq I_1 \setminus \bigcup \mathbf{J}^{\mu_1}\} \right) (\mu)} & (\mathsf{TC} \text{ def.}) \\
&= \overline{\llbracket \text{OPTIONAL}(P_1, P_2, F) \rrbracket_i^\xi}(\mu). & (\llbracket \cdot \rrbracket_i^\xi \text{ def.})
\end{aligned}$$

where $(\dagger)$ holds because $I' = \bigcup (2^{I'} \cap \mathbb{I})$, for any interval I' (in particular for $I' = I_1 \setminus \bigcup \mathbf{J}^\mu$).

■ $P = \text{PROJECT}(P_1, L)$.

$$\begin{aligned}
\mathsf{C} \left(\llbracket \text{PROJECT}(P_1, L) \rrbracket_a^\xi(\mu) \right) &= \mathsf{C} \left(\bigcup_{\mu'_L = \mu} \llbracket P_1 \rrbracket_a^\xi(\mu') \right) & (\llbracket \cdot \rrbracket_a^\xi \text{ def.}) \\
&= \mathsf{C} \left(\bigcup_{\mu'_L = \mu} \bigcup \mathsf{C} \left(\llbracket P_1 \rrbracket_a^\xi(\mu') \right) \right) & (s = \bigcup \mathsf{C}(s)) \\
&= \mathsf{C} \left(\bigcup_{\mu'_L = \mu} \bigcup \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu') \right) & (\text{IH}) \\
&= \mathsf{C} \left(\bigcup \{I \mid (I, \mu') \in \llbracket P_1 \rrbracket_i^\xi, \mu'_L = \mu\} \right) & (\bar{\cdot} \text{ def.}) \\
&= \overline{\mathsf{TC} \{ (I, \mu'') \mid (I, \mu') \in \llbracket P_1 \rrbracket_i^\xi, \mu'_L = \mu'' \}}(\mu) & (\mathsf{TC} \text{ def.}) \\
&= \overline{\mathsf{TC} \{ (I, \mu'_L) \mid (I, \mu') \in \llbracket P_1 \rrbracket_i^\xi \}}(\mu) & (\text{eliminate } \mu'') \\
&= \overline{\llbracket \text{PROJECT}(P_1, L) \rrbracket_i^\xi}(\mu). & (\llbracket \cdot \rrbracket_i^\xi \text{ def.})
\end{aligned}$$

■ $P = \text{EXTEND}(P_1, v, T)$.

- If $v \notin \text{dom}(\mu)$, then, by definition, $\overline{\llbracket P \rrbracket_i^\xi}(\mu) = \emptyset$ and $\llbracket P \rrbracket_a^\xi(\mu) = \emptyset$.
- Otherwise, let $\mu' = \mu|_{\text{dom}(\mu) \setminus \{v\}}$.

$$\begin{aligned}
\mathsf{C} \left(\llbracket \text{EXTEND}(P_1, v, T) \rrbracket_a^\xi(\mu) \right) &= \mathsf{C} \left(\llbracket P_1 \rrbracket_a^\xi(\mu') \right) & (\llbracket \cdot \rrbracket_a^\xi \text{ def.}) \\
&= \mathsf{C} \left(\bigcup \mathsf{C} \left(\llbracket P_1 \rrbracket_a^\xi(\mu') \right) \right) & (s = \bigcup \mathsf{C}(s)) \\
&= \mathsf{C} \left(\bigcup \overline{\llbracket P_1 \rrbracket_i^\xi}(\mu') \right) & (\text{IH}) \\
&= \mathsf{C} \left(\bigcup \{I \mid (I, \mu') \in \llbracket P_1 \rrbracket_i^\xi\} \right) & (\bar{\cdot} \text{ def.}) \\
&= \mathsf{C} \left(\bigcup \{I \mid (I, \mu') \in \llbracket P_1 \rrbracket_i^\xi, \mu = \mu' \dot{\cup} \{v \mapsto \llbracket T \rrbracket^{\mu'}\}\} \right) & (\mu' \text{ def.}) \\
&= \overline{\mathsf{TC} \{ (I, \mu'') \mid (I, \mu') \in \llbracket P_1 \rrbracket_i^\xi, \mu'' = \mu' \dot{\cup} \{v \mapsto \llbracket T \rrbracket^{\mu'}\}\}}(\mu) & (\mathsf{TC} \text{ def.}) \\
&= \overline{\mathsf{TC} \{ (I, \mu' \dot{\cup} \{v \mapsto \llbracket T \rrbracket^{\mu'}\}) \mid (I, \mu') \in \llbracket P_1 \rrbracket_i^\xi \}}(\mu) & (\text{eliminate } \mu'') \\
&= \overline{\llbracket \text{EXTEND}(P_1, v, T) \rrbracket_i^\xi}(\mu). & (\llbracket \cdot \rrbracket_i^\xi \text{ def.})
\end{aligned}$$

■ $F = \text{EXISTS}(P)$. Since $\llbracket \text{EXISTS}(P) \rrbracket_a^\xi$ is either $\top$ or $\perp$, the following proves the claim:

$$\begin{aligned}
\llbracket \text{EXISTS}(P) \rrbracket_a^\xi &= \top \text{ iff } \llbracket P \rrbracket_a^\xi(\mu) \neq \emptyset, \text{ for some } \mu, & (\llbracket \cdot \rrbracket_a^\xi \text{ def.}) \\
&\text{ iff } \bigcup \mathsf{C} \left(\llbracket P \rrbracket_a^\xi(\mu) \right) \neq \emptyset, \text{ for some } \mu, & (s = \bigcup \mathsf{C}(s))
\end{aligned}$$

- iff $\bigcup \overline{\llbracket P \rrbracket_i^\xi}(\mu) \neq \emptyset$, for some μ , (IH)
- iff $\bigcup \{I \mid (I, \mu) \in \llbracket P \rrbracket_i^\xi\} \neq \emptyset$, for some μ , ($\neg$ def.)
- iff $\llbracket P \rrbracket_i^\xi \neq \emptyset$ ($\emptyset \notin \mathbb{I}$)
- iff $\llbracket \text{EXISTS}(P) \rrbracket_i^\xi = \top$. ($\llbracket \cdot \rrbracket_i^\xi$ def.)

- $F \neq \text{EXISTS}(P)$. For all other types of filter expressions, $\llbracket F \rrbracket_i^\xi = \llbracket F \rrbracket_a^\xi$ trivially holds by induction on the structure of F , because the definitions of $\llbracket F \rrbracket_i^\xi$ and $\llbracket F \rrbracket_a^\xi$ are analogous.

B Experiments

- **Listing 6** SQL translation of the Flights Query 1 in Listing 4.

```
SELECT DISTINCT GREATEST(f1.ad,f2.ad) AS ad, LEAST(f1.aa,f2.aa) AS aa, f1.id
FROM flights f1 JOIN flights f2
ON GREATEST(f1.ad,f2.ad) < LEAST(f1.aa,f2.aa) AND f1.fap='FRA' AND f2.fap='MUC'
```

- **Listing 7** SQL translation of the Flights Query 2 in Listing 5.

```
WITH
-- name left- and right-hand-side arguments of the left join
lefty AS (
  SELECT GREATEST(f1.ad,f2.ad) AS ad, LEAST(f1.aa,f2.aa) AS aa,
         f1.id AS id1, f1.fap AS fap1, f2.fap AS fap2, ROW_NUMBER() OVER () AS rownum
  FROM flights f1, flights f2
  WHERE GREATEST(f1.ad,f2.ad) < LEAST(f1.aa,f2.aa) AND f1.fap='FRA' AND f2.fap='MUC'),
righty AS (
  SELECT aa, ad, id, fap, ROW_NUMBER() OVER () AS rownum
  FROM flights
  WHERE fap='CDG'),

-- gather interval split points (and remove duplicates)
split_points AS (SELECT ad AS t FROM lefty UNION SELECT aa AS t FROM lefty
                  UNION SELECT ad AS t FROM righty UNION SELECT aa AS t FROM righty),

-- split the left-hand side with the interval split points
left_split AS (
  -- interval: current ad to next ad (or lastaa if it's the last interval in the group)
  SELECT ad, COALESCE(LEAD(ad) OVER w, lastaa) AS aa, id1, fap1, fap2
  FROM (
    SELECT rownum, ad, aa AS lastaa, id1, fap1, fap2
    FROM lefty
    UNION ALL
    SELECT rownum, t AS ad, aa AS lastaa, id1, fap1, fap2
    FROM lefty JOIN split_points ON (t > ad AND t < aa))
  WINDOW w AS (PARTITION BY rownum ORDER BY ad NULLS FIRST)),

-- split the right-hand side with the interval split points
right_split AS (
  SELECT ad, COALESCE(LEAD(ad) OVER w, lastaa) AS aa, id, fap
  FROM (
    SELECT rownum, ad, aa AS lastaa, id, fap
    FROM righty
    UNION ALL
    SELECT rownum, t AS ad, aa AS lastaa, id, fap
    FROM righty JOIN split_points ON (t > ad AND t < aa))
  WINDOW w AS (PARTITION BY rownum ORDER BY ad NULLS FIRST))

-- now left_split and right_split refer to the same interval endpoints
SELECT DISTINCT ls.id1, rs.id, ls.ad, ls.aa
FROM left_split ls LEFT JOIN right_split rs ON (ls.ad = rs.ad AND ls.aa = rs.aa)
```