Spatial Ontology-Mediated Query Answering over Mobility Streams

Thomas Eiter[1], Josiane Xavier Parreira[2], and Patrik Schneider[1,2]

[1] Vienna University of Technology, Vienna, Austria
[2] Siemens AG Österreich, Vienna, Austria

Abstract. The development of (semi)-autonomous vehicles and communication between vehicles and infrastructure (V2X) will aid to improve road safety by identifying dangerous traffic scenes. A key to this is the Local Dynamic Map (LDM), which acts as an integration platform for static, semi-static, and dynamic information about traffic in a geographical context. At present, the LDM approach is purely database-oriented with simple query capabilities, while an elaborate domain model as captured by an ontology and queries over data streams that allow for semantic concepts and spatial relationships are still missing. To fill this gap, we present an approach in the context of ontology-mediated query answering that features conjunctive queries over DL-Lite$_A$ ontologies allowing spatial relations and window operators over streams having a pulse. For query evaluation, we present a rewriting approach to ordinary DL-Lite$_A$ that transforms spatial relations involving epistemic aggregate queries and uses a decomposition approach that generates a query execution plan. Finally, we report on experiments with two scenarios and evaluate our implementation based on the stream RDBMS PipelineDB.

1 Introduction

The development of (semi)-autonomous vehicles needs extensive communication between vehicles and the infrastructure, which is covered by Cooperative Intelligent Transport Systems (C-ITS). These systems collect temporal data (e.g., traffic light signal phases) and geospatial data (e.g., GPS positions), which is exchanged by vehicle-to-vehicle, vehicle-to-infrastructure, and combined communications (V2X). V2X aids to improve road safety by analyzing traffic scenes that could lead to accidents (e.g. red light violation). A key technology for this is the Local Dynamic Map (LDM) [1], which acts as an integration platform for static, semi-static, and dynamic information in a geographical context.

Current approaches for an LDM, however, are purely database-oriented with simple query capabilities. Our aim is to enable spatial-stream conjunctive queries (CQs) over a semantically enriched LDM for safety applications, such as detection of red light violations on complex intersections managed by a roadside C-ITS station. To realize spatial query answering (QA) over mobility streams, spatial and streaming data must be lifted to the setting of ontology-mediated QA with the frequently used ontology language DL-Lite$_A$. However, bridging the gap between stream processing and ontology-mediated QA is not straightforward, as the semantics of DL-Lite$_A$ must be extended with spatial relations and stream queries using window operators. For this, we build on the work on spatial QA in [12] and extend ontology-mediated QA with epistemic aggregate queries (EAQs) [10] to detemporalize the streams. The extension preserves first-order rewritability, which allows us to evaluate a CQ with spatial atoms over a stream RDBMS. Our contributions are briefly summarized as follows:

- we outline the field of V2X integration using LDMs in the mobility context (Sec. 2);
- we introduce a data model and query language suited for mobility streams (Secs. 3, 4);
- we present a spatial-stream QA approach for DL-Lite$_A$ defining its semantics with the focus of preserving FO-rewritability. The QA approach is based on CQ over DL-Lite$_A$ ontologies, which combines window operators over streams having a pulse and spatial relations over spatial objects (Secs. 4 and 5);
- we provide a technique for query rewriting taking the above into account. For query evaluation, we extend and apply the known techniques of (a) epistemic aggregate queries, e.g., average, for a "detemporalization" of the streams; and (b) provide a technique for query decomposition using hypertrees (Secs. 5 and 6);
- we have implemented a prototype and performed experiments in two scenarios to evaluate its applicability (Sec. 7).

In the final Section 8, we discuss related work and conclude with ongoing and future work.

2 V2X Integration using a Local Dynamic Map

The base communication technologies (i.e., the IEEE 802.11p standard) allow wireless access in vehicular environments, which enables messaging between vehicles themselves and the infrastructure, called V2X communication. Traffic participants and roadside C-ITS stations broadcast every 100ms messages for informing others about their current state such as position, speed, and traffic light signal phases [1]. The main types of V2X messages are *Cooperative Awareness Messages* (CAM) that provide high frequency status updates of a vehicle's position, speed, vehicle type, etc.; *Map Data Messages* (MAP) that describe the detailed topology of an intersection, including its lanes and their connections; *Signal Phase and Timing Messages* (SPaT) that give the projected signal phases (e.g., green) for a lane; and *Decentralized Environmental Notification Messages* (DENM) that inform if specific events like road works occur in a designated area.

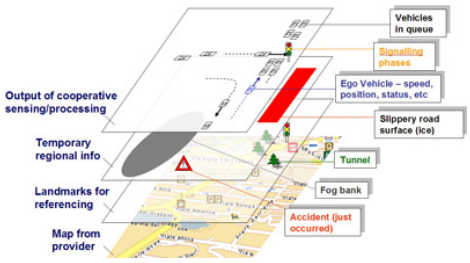

Fig. 1: The four layers of a LDM [1]

The *Local Dynamic Map* (LDM) is a comprehensive integration effort of V2X messages; the SAFESPOT project [1] introduced the concept of an LDM as an integration platform to combine static geographic information system (GIS) maps with data from dynamic environmental objects (e.g., vehicles, pedestrians). This was motivated by advanced safety applications (e.g. detect red light violation) that need an "overall" picture of the traffic environment. The LDM has the following four layers (see Fig. 1):

- *Permanent static*: the first layer contains static information obtained from GIS maps and includes roads, intersections, and points-of-interest;
- *Transient static*: the second layer extends the static map by detailed local traffic informations such as fixed ITS stations, landmarks, and intersection features like lanes;
- *Transient dynamic*: the third layer contains temporary regional information like weather, road or traffic conditions (e.g., traffic jams), and signal phases;

- *Highly dynamic*: the fourth layer contains dynamic information of road users detected by V2X messages, in-vehicle sensors like the GPS module.

Current research (e.g., [19]) on architectures of an LDM identified that it can be built on top of a spatial RDBMS enhanced with streaming capabilities. As recognized by [19], an LDM should be represented by a world model, world objects, and data sinks on the streamed input. However, an elaborate domain model, captured by an LDM ontology, and extended queries over data streams allowing spatial relations are still missing. The ontology represents an integration schema modeled in DL-Lite$_A$ and captures the layers of an LDM. Likewise, the LDM ontology must represent the content of the V2X messages and more general GIS objects (e.g., parking or petrol stations) (cf. [11])

Safety applications on intersections. "Road intersection safety" is an important application for improving road safety [1]. Intersections are the most complex environments and need special attention, where hazardous situations like *obstructed view* or *red-light violation* might lead to accidents. We take the latter as a motivation and running example.

Example 1. The following query detects red-light violations on intersections by searching for vehicles y with speed above 30km/h on lanes x whose signals will turn red in 4s:

$$\begin{aligned} q_1(x,y) : \; & LaneIn(x) \wedge hasLocation(x,u) \wedge intersects(u,v) \wedge pos_{(line,\ 4s)}(y,v) \\ & \wedge Vehicle(y) \wedge speed_{(avg,\ 4s)}(y,r) \wedge (r > 30) \wedge \ isManaged(x,z) \\ & \wedge SignalGroup(z) \wedge hasState_{(first,\ -4s)}(z, Stop) \end{aligned}$$

Query q_1 exhibits the different dimensions which need to be combined: (a) $Vehicle(y)$ and $isManaged(x,z)$ are ontology atoms, which have to be unfolded in respect to the ITS domain modelled in the LDM ontology; (b) $intersects(u,v)$ and $hasLocation(x,u)$ are spatial atoms, where the first checks spatial intersection and the second the assignment of a geometry to an object; (c) $speed_{(Avg\ 2s)}(y,v)$ defines a window operator that aggregates the average speed of the vehicles over the stream and $hasState_{(first,\ -4s)}$ gives us the upcoming traffic light state.

3 Streams, Pulses, and Spatial Databases

We now introduce the data model and sources that are used in our spatial-stream QA.

Streams and pulses. Our data model is *point-based* (vs. *interval-based*) and captures the *valid time* (vs. *transaction time*) saying that some data item is valid at that time point. We extend this validity of time, and say that a data item is valid *from its time point until the next data item is added* to the stream. To capture streaming data, we introduce the *timeline* $\mathbb{T}$, which is a *closed* interval of $(\mathbb{N}, \leq)$. A (data) *stream* is a triple $F = (\mathbb{T}, v, P)$, where $\mathbb{T}$ is a timeline, $v : \ \mathbb{T} \rightarrow \langle \mathcal{F}, \mathcal{S}_\mathcal{F} \rangle$ is a function that assigns to each element of $\mathbb{T}$, called *timestamp* (or time point), data items (called *membership assertions*) of $\langle \mathcal{F}, \mathcal{S}_\mathcal{F} \rangle$, where $\mathcal{F}$ (resp. $\mathcal{S}_\mathcal{F}$) is a *stream (resp. spatial with streams) database*, and P is an integer called *pulse* defining the general interval of consecutive data items on the timeline (cf. [6,20]). A pulse generates a stream of data items with the frequency derived from the interval length. We always have a *main pulse* $P_\mathbb{T}$ with a fixed interval length (usually 1) that defines the lowest granularity of the validity of data items. The pulse also aligns the data items, which arrive asynchronously in the database (DB), to the timeline.

Extending [20], we allow additional *larger pulses* that generate streams with a lower frequency allowing larger intervals. Larger pulses also imply that their generated data items are valid longer than items from the main pulse, thus allowing us to resize the window size

of a query and perform optimizations such as caching. Furthermore, pull-based queries are executed at any single time point i denoted as $\mathbb{T}_i$. Push-based queries are evaluated asynchronously where the lowest granularity is given by $P_{\mathbb{T}}$.

Example 2. For the timeline $\mathbb{T} = [0, 100]$, we have the stream $F_{CAM} = (\mathbb{T}, v, 1)$ of vehicle positions and speed at the assigned time points $v(0) = \{speed(c_1, 30),\ pos(c_1, (5, 5)),\ speed(b_1, 10),\ pos(b_1, (1, 1))\}$, $v(1) = \{speed(c_1, 29),\ pos(c_1, (6, 5))\ \ speed(b_1, 5),\ pos(b_1, (2, 1))\}$, and $v(2) = \{speed(c_1, 34),\ pos(c_1, (7, 5))\}$ for the individuals c_1 and b_1. A second "slower" stream $F_{SPaT} = (\mathbb{T}, v, 5)$ captures the next signal state of a traffic light: $v(0) = \{hasState(t_1, Red)\}$ and $v(5) = \{hasState(t_1, Green)\}$. As F_{SPaT} has a pulse of $p = 5$, we know $v(4) = \emptyset$ but under an alternative semantics with an *inertia assumption*, we could conclude $v'(4) = \{hasState(t_1, Red)\}$. Further, the static ABox contains the assertions $Car(c_1)$, $Bike(b_1)$, and $SignalGroup(t_1)$.

Spatial databases and topological relations. We recall the essential idea based on *Point-Set Topological Relations* (see [12]). Spatial relations are defined via pure set theoretic operations on a point set $P_E \subseteq \mathbb{R}^2$ in the plane. An *admissible geometry* $g(s)$ is a sequence $p = (p_1, \ldots, p_n)$ of points over P_F, where $P_F \subseteq P_E$ is the set of explicit points. We define a *spatial DB* over Γ_S as a pair $\mathcal{S} = (P_F, g)$ of a point set P_F and a mapping $g : \Gamma_S \to \bigcup_{i \geq 1} {P_F}^i$, where Γ_S is a set of spatial objects. The *extent* of a geometry p (full point set) is given by the function $points(p)$ as a (possibly infinite) subset of P_E. For a spatial object s, we let $g(s)$ be its geometry and let $points(s) := points(g(s))$. For our KB, we consider the following admissible geometries p over P_F, and let $P_E = \bigcup_{s \in \Gamma_S} points(s)$ (see [12] for further ones):
- *points* are the sequences $p = (p_1)$, where $points(p_1) = \{p_1\}$;
- *line segments* are sequences $p = (p_1, p_2)$, and $points(p) = \{\alpha p_1 + (1 - \alpha)p_2 \,|\, \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1\}$;

We use $points$ to evaluate the spatial relations of two spatial objects via their respective geometries and define the relations in terms of *pure set operations* (see [12] for more):
- $Inside(x, y)$: $points(x) \subseteq points(y)$ and $Outside(x, y)$: $points(x) \cap points(y) = \emptyset$;
- $Contains(x, y)$: $points(y) \subseteq points(x)$, $Intersect(x, y)$: $points(x) \cap points(y) \neq \emptyset$.

A spatial relation $S(s, s')$ with $s, s' \in \Gamma_S$ *holds* on a spatial DB $\mathcal{S}$, written $\mathcal{S} \models S(s, s')$, if $S(g(s), g(s'))$ is true. Relative to $points$, this is easily captured by a first-order (FO) formula over $(\mathbb{R}^2, \leq)$, and on geo-spatial RDBMS rewritable into FO queries.

Combining spatial and stream databases. Following an ontology-mediated QA approach, the LDM ontology is the global schema called the TBox $\mathcal{T}$, whereon we link normal, spatial, and stream DBs. We distinguish between a (standard) static ABox $\mathcal{A}$, a stream ABox $\mathcal{F}$, a static-spatial ABox $\mathcal{S}_\mathcal{A}$, and a spatial ABox with stream support $\mathcal{S}_\mathcal{F}$. These ABoxes can be stored in respective DBs, and combined in different ways. We focus on a stream DB with limited support for spatial data, which acts also as a storage for $\mathcal{S}_\mathcal{A}$.

4 Syntax, Semantics, and Query Language of DL-Lite$_A$ (S,F)

We start from previous work in [12], which introduced spatial CQ answering for DL-Lite$_A$, and lift the semantics from the spatial DL-Lite$_A$ KB to the spatial-stream KB.

Syntax and semantics of DL-Lite$_A$. We consider a vocabulary of individual names Γ_I, domain values Γ_V (e.g., $\mathbb{N}$), and spatial objects Γ_S. Given atomic concepts A, atomic roles P, and atomic attributes E, we define (a) *basic* concepts B, *basic roles* Q, and *basic*

value-domains E (attribute ranges); (b) *complex* concepts C, *complex* role expressions R, and *complex attributes* V; and (c) value-domain expressions D:

$$\begin{array}{llll} B & ::= & A \mid \exists Q \mid \delta(U_C) & C ::= \top_C \mid B \mid \neg B \mid \exists Q.C' \\ E & ::= & \rho(U_C) & D ::= \top_D \mid D_1 \mid \ldots \mid D_n \\ Q & ::= & P \mid P^- & R ::= Q \mid \neg Q \qquad V ::= U \mid \neg U \end{array}$$

where P^- is the *inverse* of P, $\top_D$ is the *universal* value-domain and $\top_C$ is the *universal* concept; furthermore, U_C is a given attribute with domain $\delta(U_C)$ (resp. range $\rho(U_C)$). A DL-Lite$_A$ *knowledge base* (KB) is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ consist of finite sets of axioms as follows:

- *inclusion assertions* of the form $B \sqsubseteq C$, $Q \sqsubseteq R$, $E \sqsubseteq D$, and $U \sqsubseteq V$; respectively
- *functionality assertions* of the form $funct\ Q$ and $funct\ U$;
- *membership assertions* of the form $A(a)$, $D(c)$, $P(a, b)$, and $U(a, c)$, where a, b are individual names in Γ_I and c is a value in Γ_V.

The semantics of DL-Lite$_A$ is in terms of FO interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where the domain $\Delta^{\mathcal{I}} \neq \emptyset$ is the disjoint union of $\Delta_I^{\mathcal{I}}$ of $\Delta_V^{\mathcal{I}}$ and $\cdot^{\mathcal{I}}$ is an *interpretation function* as usual (see [9]). Satisfaction of axioms and logical implication are denoted by $\models$. We assume the *unique name assumption* (UNA) for different individuals resp. domain values and adopt the *constant domain assumption*, saying that all models share the same domain.

Syntax DL-Lite$_A$ (S,F). Let $\mathbb{T}$ be a timeline and let Γ_S, Γ_I, and Γ_V be pairwise disjoint sets as above. A *spatial-stream knowledge base* is a tuple

$$\mathcal{K}_{\mathcal{SF}} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}_{\mathcal{A}}, \langle \mathcal{F}, \mathcal{S}_{\mathcal{F}} \rangle, \mathcal{B} \rangle,$$

where $\mathcal{T}$ (resp. $\mathcal{A}$) is a DL-Lite$_A$ TBox (resp. ABox), $\mathcal{S}_{\mathcal{A}}$ is a spatial DB, and $\langle \mathcal{F}, \mathcal{S}_{\mathcal{F}} \rangle$ is a stream DB with support for spatial data. Furthermore, $\mathcal{B} \subseteq \Gamma_I \times \Gamma_S$ is a partial function called the *spatial binding* from $\mathcal{A}$ to $\mathcal{S}_{\mathcal{A}}$ and $\mathcal{F}$ to $\mathcal{S}_{\mathcal{F}}$. If we restrict to a spatial KB resp. stream KB, we drop $\mathcal{F}$ (resp., $\mathcal{S}$) and have

$$\mathcal{K}_{\mathcal{S}} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}_{\mathcal{A}}, \mathcal{B} \rangle \text{ resp. } \mathcal{K}_{\mathcal{F}} = \langle \mathcal{T}, \mathcal{A}, \mathcal{F} \rangle.$$

We introduce for DL-Lite$_A$ the possibility to specify the *localization* of atomic concepts and roles. For this, we extend their syntax similar as in [12] as follows:

$$\begin{array}{l} C ::= \top_C \mid B \mid \neg B \mid \exists Q.C' \mid \boldsymbol{(loc\ A)} \mid \boldsymbol{(loc_s\ A)} \\ R ::= Q \mid \neg Q \mid \boldsymbol{(loc\ Q)} \mid \boldsymbol{(loc_s\ Q)}, \end{array}$$

where $s \in \Gamma_S$ and the concept and roles are as before. Intuitively, $(loc\ A)$ is the set of individuals in A that can have a spatial extension (e.g., $(loc\ Parks)$), and $(loc_s\ A)$ is the subset where it is s (e.g., $(loc_{(48.20,16.37)}\ Vienna)$).

The extension with streaming is captured by the following axiom schemes:

$$\boldsymbol{(stream_F\ C)} \text{ and } \boldsymbol{(stream_F\ R)},$$

where F is a particular stream over either complex concepts C or roles R in $\langle \mathcal{F}, \mathcal{S}_{\mathcal{F}} \rangle$.

Example 3. For Ex. 2, a TBox may contain $(stream_{F_{CAM}}\ speed)$, $(stream_{F_{CAM}}\ (loc\ pos))$, $(stream_{F_{CAM}}\ Vehicle)$, and $(stream_{F_{SPaT}}\ hasState)$, and we have further axioms $Car \sqsubseteq Vehicle$, $Bike \sqsubseteq Vehicle$, and $Ambulance \sqsubseteq \exists hasRole.Emergency$.

Semantics DL-Lite$_A$ (S,F). We give a semantics to the localization $(loc\ Q)$ and $(loc_s\ Q)$ for individuals of Q with some spatial extension resp. located at s, such that a KB $\mathcal{K}_{\mathcal{S}} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{B} \rangle$ can be readily transformed into an ordinary DL-Lite$_A$ KB $\mathcal{K}_{\mathcal{O}} = \langle \mathcal{T}', \mathcal{A}' \rangle$, using the fresh spatial top concept $C_{\mathcal{S}_{\mathcal{T}}}$ and spatial concepts C_s. An *interpretation of*

$\mathcal{K}_\mathcal{S}$ is a structure $\mathcal{I}_S = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I}, b^\mathcal{I} \rangle$, where $\langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$ is an interpretation of $\langle \mathcal{T}, \mathcal{A} \rangle$ and $b^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Gamma_S$ is a partial function that assigns some individuals a location, such that for every $a \in \Gamma_I$, $(a, s) \in \mathcal{B}_\mathcal{A}$ implies $b^\mathcal{I}(a^\mathcal{I}) = s$. We extend the semantics with $(loc\ Q)$ and $(loc_s\ Q)$, where Q is an atomic role in $\mathcal{T}$ by ($(loc\ A)$ and $(loc_s\ A)$ are accordingly):

$$(loc\ Q)^{\mathcal{I}_S} \supseteq \{(a_1, a_2) \mid (a_1, a_2) \in Q^\mathcal{I} \wedge \exists s \in \Gamma_S : (a_2, s) \in b^\mathcal{I}\},$$
$$(loc_s\ Q)^{\mathcal{I}_S} = \{(a_1, a_2) \mid (a_1, a_2) \in Q^\mathcal{I} \wedge (a_2, s) \in b^\mathcal{I}\}.$$

The transformation of $\mathcal{K}_\mathcal{S}$ to an ordinary DL-Lite$_A$ KB $\mathcal{K}_\mathcal{O}$ is described in [12] and [13].

The idea of an initial streaming semantics is by interpreting the stream over the full timeline, which can be captured by a finite sequence $\mathcal{F}_\mathcal{A} = (\mathcal{F}_i)_{\mathbb{T}_{\min} \leq i \leq \mathbb{T}_{\max}}$ of temporal ABoxes, which is obtained via the evaluation function v on $\mathcal{F}$ and $\mathbb{T}$ (cf. [7], [15]). Hence, we define the interpretation of the point-based model over $\mathbb{T}$ as a sequence $\mathcal{I}_F = (\mathcal{I}_i)_{\mathbb{T}_{\min} \leq i \leq \mathbb{T}_{\max}}$ of interpretations $\mathcal{I}_i = \langle \Delta^\mathcal{I}, \cdot^{\mathcal{I}_i} \rangle$; $\mathcal{I}_F$ is a model of $\mathcal{K}_\mathcal{F}$, denoted $\mathcal{I}_F \models \mathcal{K}_\mathcal{F}$ iff $\mathcal{I}_i \models \mathcal{F}_i$ and $\mathcal{I}_i \models \mathcal{T}$, for all $i \in \mathbb{T}$.

The semantics of the $(stream_F\ C)$ and $(stream_F\ R)$ axioms is along the same line. A stream axiom is satisfied, if a complex concept C (resp. role R) holds over all the time points of stream $F = (\mathbb{T}, v, P)$; thus we restrict our models such that:

$$(stream_F\ C)^\mathcal{I} = \bigcap_{i \in tp(\mathbb{T},P)} C^{\mathcal{I}_i} \text{ and } (stream_F\ R)^\mathcal{I} = \bigcap_{i \in tp(\mathbb{T},P)} R^{\mathcal{I}_i},$$

where $tp(\mathbb{T}, P)$ is a set of time points determined by the segmentation of $\mathbb{T}$ by P. This allows us to check for the *satisfiability* of a KB and gives us a global consistency, which is of theoretical nature, since we would need to know the full timeline.

Spatial-stream query language over DL-Lite$_A$ (S,F). We next define spatial-stream conjunctive queries over $\mathcal{K}_{\mathcal{SF}}$. Such queries may contain ontology, spatial, and stream predicates. An spatial-stream CQ $q(\mathbf{x})$ is a formula:

$$\bigwedge_{i=1}^{l} Q_{O_i}(\mathbf{x}, \mathbf{y}) \wedge \bigwedge_{j=1}^{n} Q_{S_j}(\mathbf{x}, \mathbf{y}) \wedge \bigwedge_{k=1}^{m} Q_{F_k}(\mathbf{x}, \mathbf{y}) \quad (1)$$

where $\mathbf{x}$ are the *distinguished* (*answer*) variables, $\mathbf{y}$ are either *non-distinguished* (*existentially quantified*) variables, individuals from Γ_I, or values from Γ_V and

- each $Q_{O_i}(\mathbf{x}, \mathbf{y})$ has the form $A(z)$ or $P(z, z')$, where A is a concept name, P is a role name and z, z' are from $\mathbf{x} \cup \mathbf{y}$;
- each atom $Q_{S_j}(\mathbf{x}, \mathbf{y})$ is from the vocabulary of spatial relations (see Sec. 3) and of the form $S(z, z')$, with z, z' from $\mathbf{x} \cup \mathbf{y}$;
- $Q_{F_j}(\mathbf{x}, \mathbf{y})$ is similar to $Q_{O_i}(\mathbf{x}, \mathbf{y})$ but adds the vocabulary for stream operators, which are taken from [6] and relate to CQL operators [3]. Moreover, we have a window $\boxplus$ over a stream F_j that is derived from L (in $\mathbb{Z}^+$ for past, or in $\mathbb{Z}^-$ for future) time units resp. $\mathbb{T}_i$, and an *aggregate function* $agr \in \{count, sum, first, \ldots\}$ (see Sec. 5 for details) that is applied to the data items in the window:[3]
  - $\boxplus_T^L agr$ represents the aggregate of last/next L time units of stream F_j;
  - $\boxplus_T$ represents the current tuples of F_j with $L = 0$;
  - $\boxplus_T^O agr$: represents the aggregate of all previous L time units of F_j;

Example 4. We modify $q_1(x, y)$ of Example 1 and use the stream operators instead:

$$q_1(x, y) : LaneIn(x) \wedge hasLocation(x, u) \wedge intersects(u, v) \wedge position_{\boxplus_T^4 line}(y, v)$$
$$\wedge\ Vehicle(y) \wedge speed_{\boxplus_T^4 avg}(y, r) \wedge (r > 30) \wedge isManaged(x, z)$$
$$\wedge\ SignalGroup(z) \wedge hasState_{\boxplus_T^{-4} first}(z, Stop)$$

[3] This would be represented in CQL as `R[Range L]`, `R[Now]`, `R[Range L Slide D]`, etc.

Certain answer semantics with spatial atoms. In the streamless setting, due to the OWA, queries are evaluated over all (possibly infinitely many) models. *Certain answers* retain the tuples that are answers in all possible models. More formally, a *match* for $q(\mathbf{x})$ in an interpretation $\mathcal{I}=\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ of $\mathcal{K}$ is a function $\pi : \mathbf{x} \cup \mathbf{y} \rightarrow \Delta^{\mathcal{I}}$ such that $\pi(c) = c^{\mathcal{I}}$, for each constant c in $\mathbf{x} \cup \mathbf{y}$, and for each $i = 1, \ldots n$ and $j = 1, \ldots, m$:

(i) $\pi(z) \in A^{\mathcal{I}}$, for $Q_{O_i}(\mathbf{x}, \mathbf{y}) = A(z)$ (concept atoms);
(ii) $(\pi(z), \pi(z')) \in P^{\mathcal{I}}$, for $Q_{O_i}(\mathbf{x}, \mathbf{y}) = P(z, z')$ (role atoms); and
(iii) $\exists s, s' \in \Gamma_S : (\pi(z), s) \in b^{\mathcal{I}} \wedge (\pi(z'), s') \in b^{\mathcal{I}} \wedge \mathcal{S} \models S(s, s')$, for $Q_{S_j}(\mathbf{x}, \mathbf{y}) := S(z, z')$ (spatial atoms).

A tuple $\mathbf{c} = c_1, \ldots, c_k$ over Γ_I is a *(certain) answer* for $q(\mathbf{x})$ in $\mathcal{I}$, $\mathbf{x} = x_1, \ldots, x_k$, if $q(\mathbf{x})$ has some match π in $\mathcal{I}$ where $\pi(x_i) = c_i$, $i = 1, \ldots, k$; and $\mathbf{c}$ is an answer for $q(\mathbf{x})$ over $\mathcal{K}$, if it is an answer in every model $\mathcal{I}$ of $\mathcal{K}$. The *result* $Cert(q(\mathbf{x}), \mathcal{K})$ of $q(\mathbf{x})$ over $\mathcal{K}$ is the set of all its answers. If we drop $\mathcal{T}$, we obtain a DB setting and let $Eval(q(\mathbf{x}), \mathcal{I})$ be the set of matches of $q(\mathbf{x})$ over the single model $\mathcal{I}$ of $\mathcal{A}$ under closed world assumption.

Regarding spatial atoms, as shown in [12,13] the semantic correspondence between $\mathcal{K}_{\mathcal{O}}$ and $\mathcal{K}_{\mathcal{S}}$ guarantees that we can rewrite $q(\mathbf{x})$ into an equivalent query $uq(\mathbf{x})$ over $\mathcal{K}_{\mathcal{S}}{}' = \langle \mathcal{T}', \mathcal{A}', \mathcal{S}_{\mathcal{A}} \rangle$. Using the rewriting and the semantic correspondence of $\mathcal{K}_{\mathcal{O}}$ and $\mathcal{K}_{\mathcal{S}}$, spatial atoms can be rewritten into a "standard" DL-Lite$_A$ UCQ, thus, answering spatial CQs is still FO-rewritable (details in [12] and [13]).

5 Query Rewriting by Stream Aggregation

We aim at answering queries at *a single* time point $\mathbb{T}_i$ with stream atoms that define *aggregate functions* on different windows sizes relative to $\mathbb{T}_i$. For this, we consider a semantics based on *epistemic aggregate queries* (EAQ) over ontologies [10] by dropping the order of time points for the membership assertions and handle the (streamed) assertions as *bags*, which is similar to "classic" stream processing approaches.

Epistemic aggregate queries. As described in [10], EAQ are defined over bags of numeric and symbolic values, called *groups* and denoted as $\{| \cdot |\}$. Aggregates cannot be directly transferred to DL-Lite, since with the certain answer semantics each model has different groups due to unknown individuals, which leads to empty answers. [10] extended database semantics for aggregates with an epistemic operator $\mathbf{K}$ and a two-layer evaluation using the completion w.r.t $\mathcal{T}$. The basic idea is to close the aggregate query, so only known individuals are grouped and aggregated. More formally, an EAQ is defined as [4]

$$q_a(\mathbf{x}, agr(y)) : \mathbf{K}\ \mathbf{x}, y, \mathbf{z}.\ \phi,$$

where $\mathbf{x}$ are the *grouping variables*, $agr(y)$ is the *aggregate function and variable*, and ϕ is a CQ called *main conditions*; $\mathbf{z}$ are the disjoint existential variables of ϕ. We call $\mathbf{w} := \mathbf{x} \cup y \cup \mathbf{z}$ the $\mathbf{K}$-variables of ϕ. The definition of a group was extended in [10] by a multiset $H_{\mathbf{d}}$ of groups $\mathbf{d}$, called $\mathbf{K}$-group, as:

$$H_{\mathbf{d}} := \{| \ \pi(y) \mid \pi \in KSat_{\mathcal{I},\mathcal{K}}(\mathbf{z}; \phi) \text{ and } \pi(\mathbf{x}) = \mathbf{d} \ |\},$$

where $KSat$ are the *satisfying* $\mathbf{K}$-matches of ϕ *for* the model $\mathcal{I}$ of $\mathcal{K}$ and given by:

$$KSat_{\mathcal{I},\mathcal{K}}(\mathbf{w}; \phi) := \{\pi \in Eval(\phi, \mathcal{I}) \mid \pi(\mathbf{w}) \in Cert(aux_{q_a}, \mathcal{K})\},$$

where $aux_{q_a}(\mathbf{w}) \leftarrow \phi$ is the auxiliary atom used to map $\mathbf{w}$ only to *known solutions*. The set of $\mathbf{K}$-answers for an EAQ query q over $\mathcal{I}$ and $\mathcal{K}$ can now be derived as:

[4] We simplified EAQs of [10] by omitting ψ and consider only aggregates with a single variable.

$$q_a^{\mathcal{I}} := \{(\mathbf{d}, agr(H_{\mathbf{d}})) \mid \mathbf{d} = \pi(\mathbf{x}), \text{ for some } \pi \in KSat_{\mathcal{I},\mathcal{K}}(\mathbf{w}; \phi)\}.$$

The *epistemic certain answers* $ECert(q_a, \mathcal{K})$ for a query q_a over $\mathcal{K}$ is the set of $\mathbf{K}$-answers that are answers in every model $\mathcal{I}$ of $\mathcal{K}$. To compute $ECert(q_a, \mathcal{K})$, [10] gave a "general algorithm" `GA` that (1) computes the certain answers, (2) projects on the $\mathbf{K}$-variables, and (3) aggregates the resulting tuples. Importantly, evaluating EAQs reduces to standard CQ evaluation over DL-Lite$_A$ with LOGSPACE data complexity.

Filtered and merged temporal ABoxes. Our approach is to evaluate the EAQ over one or more *filtered and merged temporal* ABoxes. The filtering and merging, relative to the window size and $\mathbb{T}_i$, creates several *windowed* ABoxes $\mathcal{A}_{\boxplus_\phi}$, which are the union of the static ABox $\mathcal{A}$ and the filtered stream ABoxes from $\mathcal{F}$. The EAQ aggregates are applied on each *windowed* ABox $\mathcal{A}_{\boxplus_\phi}$ by aggregating normal objects, concrete values, and spatial objects. More formally, a stream atom $\phi \boxplus_T^L agr$ is evaluated as an EAQ over ontologies

$$q_\phi(\mathbf{x}, agr(y)) : \mathbf{K}\ \mathbf{x}, y, \mathbf{z}.\ \phi \boxplus_T^L,$$

where $\mathbf{x}$ are the grouping variables and y is the aggregate variable, $\mathbf{z}$ are the disjoint existential variables, and ϕ is a subquery of q with atoms in the same *scope* of the window operator $\boxplus_T^L$ and aggregate functions agr.

Example 5. For query $q_1(x, y)$ of Ex. 4, we have three EAQs represented as:

$$q_{pos}(y, line(v)) : \mathbf{K}\ y, v.\ Vehicle(y) \wedge position(y, v);$$
$$q_{speed}(y, avg(r)) : \mathbf{K}\ y, r.\ Vehicle(y) \wedge speed(y, r);$$
$$q_{state}(z, first(m)) : \mathbf{K}\ z, m.\ hasState(z, m)$$

We extend the evaluation of EAQs for the stream setting, such that an EAQ is evaluated over the window relative to $\mathbb{T}_i$, the window operator $\boxplus_T^L$, and the pulse P. $KSat_{\mathcal{I}_\boxplus,\mathcal{K}_\boxplus}(\mathbf{w}; \phi)$ is now the set of $\mathbf{K}$-matches of ϕ for a model $\mathcal{I}_\boxplus$ of $\mathcal{K}_\boxplus$, where the *windowed* ABox $\mathcal{A}_\boxplus$ is defined as $\mathcal{A}_\boxplus = \mathcal{A} \cup \bigcup\{\mathcal{A}_i \mid w_s \leq i \leq w_e\}$. We have four cases for the window size L and a pulse P, where P enlarges L according to its interval length:

- a current window with $L = 0$, i.e. $w_s = w_e = \mathbb{T}_i$;
- a past window with $L > 0$ leading to $w_s = (\mathbb{T}_i - L)$ and $w_e = \mathbb{T}_i$;
- a future window with $L < 0$ that is $w_s = \mathbb{T}_i$ and $w_e = (\mathbb{T}_i + L)$; and
- the entire history with O resulting in $w_s = 0$ and $w_e = \mathbb{T}_i$.

We obtain KB $\mathcal{K}_\boxplus = \langle \mathcal{T}, \mathcal{A}_\boxplus \rangle$ as above; the *epistemic (certain) answers* for q_ϕ over $\mathcal{K}_\boxplus$ are naturally defined as $ECert_\boxplus(q_\phi, \mathcal{K}_\boxplus) = \bigcap_{\mathcal{I}_\boxplus \models \mathcal{K}_\boxplus} q_\phi^{\mathcal{I}_\boxplus}$, where

$$q_\phi^{\mathcal{I}_\boxplus} = \{(\mathbf{d}, agr(H_{\mathbf{d}})) \mid \mathbf{d} = \pi(\mathbf{x}), \text{ for some } \pi \in KSat_{\mathcal{I}_\boxplus,\mathcal{K}_\boxplus}(\mathbf{w}; \phi)\}$$

are the $\mathbf{K}$-matches that are answers in the model $\mathcal{I}_\boxplus$ of $\mathcal{K}_\boxplus$. In $ECert_\boxplus$, we did not yet address the *validity* of an assertion, say in $\mathcal{A}_{\boxplus_1}$, until the next assertion in $\mathcal{A}_{\boxplus_3}$. Two semantics are suggestive: the first ignores intermediate time points, and thus $\mathcal{A}_{\boxplus_2}$ will be unknown. The second fills the missing gaps with the previous assertion, i.e. copies it from $\mathcal{A}_{\boxplus_1}$ to $\mathcal{A}_{\boxplus_2}$. For specific aggregate functions, e.g., max, min, or $last$, the two semantics coincide, but for sum, avg, and $count$, they are different.

Example 6. We pose the query $q_1(x, y)$ at $\mathbb{T}_1$ and replace the stream atoms with auxiliary atoms related to the EAQ of Ex. 5:

$$q_1(x, y) : LaneIn(x) \wedge hasLocation(x, u) \wedge intersects(u, v) \wedge q_{pos}(y, v)$$
$$\wedge q_{speed}(y, r) \wedge (r > 30) \wedge isManaged(x, z) \wedge q_{state}(z, Stop)$$

The queries are computed using the ABoxes $\mathcal{A}_{\boxplus[0,1]} = \mathcal{A} \cup \mathcal{A}_0 \cup \mathcal{A}_1$ and $\mathcal{A}_{\boxplus[1,4]} = \mathcal{A} \bigcup_{1 \leq i \leq 4} \mathcal{A}_i$. This leads under $ECert_\boxplus$ for q_{speed} to the groups G_{c_1}=$\{|30, 29, 34|\}$

Algorithm 1: NSQ - Answer Naive Stream Query

Input: A stream conjunctive query q, time point $\mathbb{T}_i$, and a KB $\mathcal{K}_\mathcal{F}$
Output: Set of tuples O
/* Step 1: Detemporalize */
foreach Q_{F_i} *of* q **do**
  $\mathcal{A}_{\boxplus_i} \leftarrow \mathcal{A} \bigcup_{w_s \leq j \leq w_e} \mathcal{A}_j$ according to $\boxplus_T^L$ and $\mathbb{T}_i$;
  $\mathcal{K}_{\boxplus_i} \leftarrow \langle \mathcal{T}, \mathcal{A}_{\boxplus_i} \rangle$;
  build $aux_i(\mathbf{x}, y, \mathbf{z})$ from $\phi \boxplus_T^L agr$ of Q_{F_i} ;
  build $q_{i,1}(\mathbf{x}, y, \mathbf{z}^\phi)$ from PerfectRef$(aux_i, \mathcal{T})(\mathbf{x}, y, \mathbf{z})$;
  build $q_{i,2}(x, agr(y))$ from $q_{i,1}(\mathbf{x}, y, \mathbf{z}^\phi)$ and $\phi \boxplus_T^L agr$;
  $R_{i,1}$:= evaluate Answer$(aux_i, \mathcal{K}_{\boxplus_i})$ /* certain answers */ ;
  $R_{i,2}$:= evaluate $q_{i,1}$ over $R_{i,1}$ /* $\mathbf{K}$ projection */ ;
  R_i := evaluate $q_{i,2}$ over $R_{i,2}$ /* aggregation */ ;
  Add R_i to $\mathcal{A}_{aux}$ and replace Q_{F_i} in q with $R_i(x, y)$;
/* Step 2: Standard evaluation */
O := evaluate Answer$(q, \langle \mathcal{T}, \mathcal{A} \cup \mathcal{A}_{aux} \rangle)$;

and G_{b_1}=$\{|10, 5|\}$, which results in q_{speed}=$\{(c_1, 31), (b_1, 7.5)\}$. The results for the other EAQ are q_{state}=$\{(t_1, Red)\}$ and q_{pos}=$\{(c_1, ((5, 5), (6, 5), (7, 5))), (b_1, ((1, 1), (2, 1)))\}$.

$ECert_\boxplus$ gives the certain answers for a single EAQ including the ontology atoms in the same scope as the stream atoms. Answering the full CQ q can be done by answering each EAQ q_{ϕ_k} separately and joining the answers, i.e.,

$$ECertAll(q, \mathcal{K}_\mathcal{F}, \mathbb{T}_i) = ECert_\boxplus(q_{\phi_1}, \mathcal{K}_{\boxplus_{w_1}}) \bowtie \cdots \bowtie ECert_\boxplus(q_{\phi_j}, \mathcal{K}_{\boxplus_{w_j}}),$$

where the $w_k = w(\phi_k, \mathbb{T}_i)$ are the computed window sizes and $A \bowtie B = \{t$ over $sig(A) \cup sig(B) \mid t[sig(A)] \in A,\ t[sig(B)] \in B\}$ is the join (cf. [18]) of sets A, B of $\mathbf{K}$-answers, where $sig()$ is the relational signature of a $\mathbf{K}$-answer set. The new $\mathbf{K}$-answers are also answers in every model $\mathcal{I}$ of $\mathcal{K}$. More details on deriving the q_{ϕ_j} are given in Sec. 6.

We now introduce the algorithm NSQ (see Alg. 1), where $\mathbf{z}^\phi$ are the non-distinguished variables in ϕ and PerfectRef (resp. Answer) is the "standard" query rewriting (resp. evaluation) as in [9]. NSQ extends the GA of [10] to compute the answers for stream CQs as follows: (1) calculate the epistemic answer for each stream atom over the different windowed ABoxes and store the result in an *auxiliary ABox* using new atoms. Furthermore, replace each stream atom with a new auxiliary atom; (2) calculate the certain answers over $\mathcal{A}$ and the auxiliary ABox, using "standard" DL-Lite$_A$ query evaluation. A proof sketch for correctness of NSQ is given in [13], viz. that for every stream CQ q, KB $\mathcal{K}_\mathcal{F}$, and time point $\mathbb{T}_i$, we have NSQ$(q, \mathcal{K}_\mathcal{F}, \mathbb{T}_i) = ECertAll(q, \mathcal{K}_\mathcal{F}, \mathbb{T}_i)$. It considers that q must be constrained by $\mathcal{T}$ and that aggregate functions must obey conditions as in [10]; it exploits that answering each EAQ (Step 1) can be decoupled from answering the full CQ.

Standard aggregates. Different aggregate functions for use in $ECert(q, \mathcal{K})$ were already discussed in [10]. For $last$ and $first$, we extend the definition of $H_\mathbf{d}$, as the sequence of time points is lost. By iteratively checking if we have a match in one of the ABoxes $\mathcal{A}_{\boxplus_i} w_s \leq i \leq w_e$, we can determine the first resp. last match. The extension of $H_\mathbf{d}$ for $first$ and $last$ is by checking each model for match (details in [13]). In an implementation, the first/ last match can be simply cached while processing the stream.

Spatial aggregates. For *spatial objects*, we define geometric aggregate functions on the multiset of $H_\mathbf{d}$. As the order of assertions (i.e., points) is lost, we need to rearrange them to create an admissible geometry $g(s)$ that is a sequence $p = (p_1, \ldots, p_n)$. We add new aggregates on $H_\mathbf{d}$ to create new admissible geometries $g(s_\mathbf{d})$:

- agr_{point}: we evaluate $last$ to get the last available position p_1 and set $g(s_{\mathbf{d}}) := (p_1)$;
- agr_{line}: we create $p = (p_1, \ldots, p_n)$, where $p_1 \neq p_n$ and determine a total order on the bag of points in each $\mathbf{K}$-group, such that we have a starting point using last and iterate backwards finding the next point.
- agr_{angle}: This aggregate function determines angles (in degrees) in a geometry by applying (1) agr_{line}, then (2) obtain a simplified geometry using smoothing, and (3) calculate the angles between the lines of the geometry.

Besides the above aggregate functions, more functions such as computing the convex hull or minimum spanning tree can be applied. In contrast to numerical aggregates, spatial aggregates introduce for each $\mathbf{K}$-group $(\mathbf{d}, agr(H_{\mathbf{d}}))$ a new spatial object $s_{\mathbf{d}}$ of Γ_S and an admissible geometry $g(s_{\mathbf{d}})$ with $agr(H_{\mathbf{d}}) = (p_1, \ldots, p_n)$. This is achieved by (a) adding a binding $(\mathbf{d}, s_{\mathbf{d}})$ to $\mathcal{B}$ and (b) creating a new mapping $g : s_{\mathbf{d}} \rightarrow (p_1, \ldots, p_n)$ in $\mathcal{S}_{aux}$. For simplicity, we assume that Γ_S is static and contains (candidates for) $s_{\mathbf{d}}$ already.

Combining spatial and stream queries. We combine the spatial and temporal elements of a query q and KB $\mathcal{K}$ as follows: (1) detemporalize the stream atoms using EAQs; (2) transform q and $\mathcal{K}$ to an ordinary UCQ and KB as in Sec. 4, where in Step 2 of Alg. 1 $Cert(q, \langle \mathcal{T}, \mathcal{A} \cup \mathcal{A}_{aux}, \mathcal{S}_{\mathcal{A}} \cup \mathcal{S}_{aux}, \mathcal{B} \rangle)$ is changed to $Cert_S(uq, \langle \mathcal{T}', \mathcal{A}' \cup \mathcal{A}_{aux} \rangle)$. We still keep LOGSPACE data complexity, which follows from the data complexity of single EAQs and the fact that the number of aggregate atoms bounds the number of EAQs. As shown before, spatial binding and relations do not increase the data complexity.

6 Query Evaluation by Hypertree Decomposition

We focus on pull-based evaluation of spatial-stream CQs, which is already challenging, as we must deal with three different types of query atoms that need different evaluation techniques over possibly separate DBs. Ontology atoms are evaluated over the static ABox $\mathcal{A}$ using the "standard" DL-Lite$_A$ query rewriting, i.e., $PerfectRef$ [9]. For spatial query atoms, we need to dereference the bindings by joining the binding $\mathcal{B}$ and the spatial ABox $\mathcal{S}_{\mathcal{A}}$, where we evaluate the spatial relations (e.g., $Inside$) over the spatial objects of the join; Stream query atoms are computed as described in Alg. 1 over the stream ABox $\mathcal{F}$ and the spatial ABox with stream support $\mathcal{S}_{\mathcal{F}}$.

Evaluation strategies. In [12], we introduced spatial CQ evaluation based on the assumptions that *no bounded variables* occur in spatial atoms and the CQ $q_S(\mathbf{x})$ has to be *acyclic*. This allows an evaluation in two stages:

(1) evaluate the ontology part of $q_S(\mathbf{x})$ by dropping all spatial atoms over $\mathcal{K}_{\mathcal{S}}{}'$. For this, we can apply the standard query rewriting and evaluate the resulting UCQ over $\mathcal{A}$;
(2) filter the result of Step (1), by evaluating the spatial atoms on the matches π (for the distinguished variables $\mathbf{x}$) taking the bindings $\mathcal{B}$ to $\mathcal{S}_{\mathcal{A}}$ into account.

As shown in [12], one evaluation strategy is based on the hypergraph of q_S and the derived join plan, while another is based on compiling $q_S(\mathbf{x})$ into a single, large UCQ with spatial joins. The hypergraph-based strategy is well suited for lifting to spatial-stream CQs as the partial EAQ results are already stored. Hence, we merge it with the two-stage evaluation of Sec. 5 (detemporalization). For this, we aim to find large subqueries of combined stream and ontology atoms, and an efficient evaluation order (the join plan), which allows the partial evaluation and merging of the intermediate results to obtain the final result. In our opinion, the hypergraph-based strategy has the advantage of allowing fine-grained caching, full control over the evaluation, and possibly different DBs.

Hypergraphs and join trees. Many works have been dedicated to connecting hypergraphs, (acyclic) DB schemes, and join trees (see [18] for an overview). For decomposing a query q, the query hypergraph $H(q) = (V, E)$ is popular, where the vertices V represent the variables in q and the hyperedges in E capture the atoms in q with shared variables. In case of an *acyclic conjunctive query* (ACQ), which is defined in terms acyclicity of $H(q)$, a *join tree* can be generated from $H(q)$ that yields a plan for computing the query q. We focus here on α-acyclicity, which can be efficiently tested by the *GYO-reduction* (cf. [18]). A specific join tree J_H can be found via the *maximum-weight spanning tree* T_S of the *intersection graph* I_H of H, where edge weights of T_S are edge counts of V in I_H.

Details on the query evaluation. The combined evaluation extends our spatial evaluation strategy with hypertree decomposition of a hypergraph, by keeping intermediate results of each step in memory. The main steps of our query evaluation algorithm are:

(1) construct the α-acyclic hypergraph H_q from q and label each hyperedge in H_q with l_O, l_S, and l_F if it represents an ontology, spatial, or stream atom, resp. the combination of them; l_F gets the window size assigned, e.g., $l_{F,2}$ for $speed_{\boxplus^2_T avg}$.
(2) build the join tree J_q of H_q and extract the subtrees J_{ϕ_i} in H_q, such that each node is covered by the same label $l_{F,n}$. The intention is to extract subtree CQs that share the same window size L (where static queries have $L = 0$), so they can be evaluated together and cached for future query evaluations.
(3) apply detemporalization as in Alg. 1, where for each subtree J_{ϕ_i} the stream CQ q_{ϕ_i} is extracted and computed. The results are stored in a (virtual) relation R_{ϕ_i}, and each J_{ϕ_i} is replaced with a query atom pointing to R_ϕ.
(4) traverse J_q bottom up, left-to-right, to evaluate the CQ q_{ϕ_i} for each subtree J_{ϕ_i} (now without stream atoms) and keep the results in memory for future steps. Ontology and spatial atoms are evaluated as described before.

Example 7. The subqueries and join order of query $q3(x, v)$ in Table 1 is as follows:

(1) $q_{3,F1}(x, y) : Vehicle(x) \wedge position_{\boxplus^{10}_T line}(x, y)$;
(2) $q_{3,N1}(v, u) : LaneIn(v) \wedge hasLocation(v, u)$; and
(3) $q_3(x, v) : q_{3,F1}(x, y) \wedge intersects(y, u) \wedge q_{3,N1}(v, u)$.

Caching for future queries is achieved by storing the intermedia results in memory with an expiration time according to L and the pulse. Static results never expire.

7 Implementation and Experimental Evaluation

We have implemented a prototype for our spatial-stream QA approach in JAVA 1.8 using the open-source PIPELINEDB 9.6.1 (https://www.pipelinedb.com/) as the spatial-stream RDBMS. The hypertree decomposition for each query is computed once using the implementation at https://www.dbai.tuwien.ac.at/proj/hypertree/. Based on it, each subquery is evaluated separately and (spatial) joined in-memory. For the FO-rewriting of DL-Lite$_A$, we used the implementation of *PerfectRef* in OWLGRES 0.1 [24] for now; more recent (and more efficient) implementations for query rewriting (e.g., [23]) are available.

Our experiment is based on two scenarios of monitoring vehicles and traffic lights (a) on a single intersection and (b) on a network of locally connected intersections, both managed by a single roadside C-ITS station. The ontology, queries (see Table 1), the experimental setup with logs, and the implementation are available on http://www.kr.tuwien.ac.at/research/projects/loctrafflog/eswc2017. We use a custom DL-Lite$_A$ LDM ontology

$q_1(x, y, z)$:	$\texttt{Car}(x), \texttt{speed}(x, y)[\texttt{avg}, 10], \texttt{vehicleMaker}(x, z), y > 30$	/* cars w/ brands, travelling above 30km/h */
$q_2(x, y)$:	$\texttt{LaneIn}(x), \texttt{hasSignal}(x, y), \texttt{SignalGroup}(y), \texttt{signalState}(y, z)[\texttt{last}, 15], z = \text{"R"}$	/* lanes and signal groups switched to red */
$q_3(x, v)$:	$\texttt{Vehicle}(x), \texttt{pos}(x, y)[\texttt{line}, 10], \texttt{inside}(y, u), \texttt{hasGeo}(v, u), \texttt{LaneIn}(v)$	/* vehicles on incoming lanes */
$q_4(x, y)$:	$\texttt{Vehicle}(x), \texttt{pos}(x, w)[\texttt{line}, 30], \texttt{intersects}(w, z), \texttt{pos}(y, z)[\texttt{line}, 30], \texttt{Car}(y)$	/* vehicles with crossed paths */
$q_5(x, y)$:	$\texttt{Vehicle}(x), \texttt{speed}(x, z)[\texttt{avg}, 15], \texttt{pos}(x, y)[\texttt{line\_angle}, 15], z > 30, y > -10, y < 10$	/* vehicles above 30km/h heading straight */
$q_6(x, y)$:	Taken from Ex. 1	/* Detection of red-light violation */
$q_7(x, z)$:	$\texttt{LaneIn}(x), \texttt{isPartof}(x, u), \texttt{Intersection}(u), u = \text{"I1"}, \texttt{hasSignal}(x, y),$ $\texttt{SignalGroup}(y), \texttt{signalState}(y, r)[\texttt{last}, 15], r = \text{"R"}, \texttt{connect}(x, q), \texttt{connect}(q, v),$ $\texttt{Lane}(v), \texttt{hasSignal}(v, z), \texttt{SignalGroup}(z), \texttt{signalState}(z, s)[\texttt{last}, 15], s = \text{"R"}$	/* synthetic, testing many ontology atoms */
$q_8(x, y)$:	$\texttt{Vehicle}(x), \texttt{pos}(x, y)[\texttt{line}, 20], \texttt{intersects}(y, u), \texttt{LaneIn}(r), \texttt{hasGeo}(r, u),$ $\texttt{intersects}(y, v), \texttt{LaneIn}(s), \texttt{hasGeo}(s, v), \texttt{intersects}(y, w), \texttt{LaneIn}(t),$ $\texttt{hasGeo}(t, w), \texttt{within}(y, z), \texttt{hasGeo}(q, z), \texttt{Intersection}(q)$	/* synthetic, testing many spatial atoms */
$q_9(x, q, r, s, t, u)$:	$\texttt{Vehicle}(x), \texttt{speed}(x, q)[\texttt{avg}, 1], \texttt{speed}(x, r)[\texttt{avg}, 5], \texttt{speed}(x, s)[\texttt{avg}, 10],$ $\texttt{speed}(x, t)[\texttt{avg}, 25], \texttt{speed}(x, u)[\texttt{avg}, 50]$	/* synthetic, testing many stream atoms */

Table 1: Benchmark queries (windows size in seconds)

with 119 concepts (with 113 inclusion assertions); 34 roles and 28 data roles (with 31 inclusion assertions). The LDM ontology models the C-ITS domain in a layered approach, separating concepts like ITS features (e.g., intersection topology), geo-features (e.g., POIs), geometries (e.g., polygon), actors (e.g., vehicles), events (e.g., accidents); and roles like partonomies (e.g., isPartOf), spatial relations, and generic roles (e.g., speed).

For (a), we have a T-shaped intersection as shown in Fig. 2 that represents a real-world deployment of a C-ITS station in Vienna. It connects two roads with 13 lanes and 3 signal groups that are linked to the lanes. We developed a synthetic data generator that simulates the movement of 10, 100, 500, 1000, 2500, and 5000 vehicles on a single intersection updating the streams averagely 50ms. This allows us to generate streams with up to 10000 data points per sec. and stream.[5] We chose random starting points and simulated linear movements on a constant pace, creating a stream of vehicle positions. We also simulated simple signal phases for each traffic light that toggle between red and green every 3 secs. The aim of this scenario is to show for simple driving patterns the scalability of our approach in the number of vehicles. For (b), we use a realistic traffic simulation of 9 intersections in a grid, developed with the microscopic traffic simulation PTV VISSIM (http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/), allowing us to simulate realistic driving behavior and signal phases. The intersection structure, driving patterns and signal phases are more complex, but the number of vehicles is lower (max. 300) than in (a), as we quickly have traffic jams. We developed an adapter to extract the actual state of each simulation step, allowing us to replay the simulation from the logs. To vary data throughput, we ran the replay with 0ms, 100ms (real-time), 250ms and 500ms delay.

Results. We conducted our experiments on a Mac OS X 10.6.8 system with an Intel Core i7 2.66GHz, 8GB of RAM, and a 500GB HDD. The average of 11 runs for query rewriting time and evaluation time was calculated, where the largest outlier was dropped. The results are shown in Table 2 presenting query type (O for ontology, F for stream, and S for spatial atoms), number of subqueries $\#Q$, size of rewritten atoms $\#A$, and t as the average evaluation time (AET) in seconds for n vehicles or the delay in ms.

The baseline spatial-stream query is q_3 for 500 vehicles, where we have a time-to-load (TOL) of 0.22s, an evaluation time for the stream (resp. ontology) atom of 0.54s (resp. 0.03s), and a spatial join time of 0.05s. Clearly, 50% of the AET is use for the stream atom (including rewriting steps). The TOL could be reduced by pre-compiling the program;

[5] The intervals vary due to the number of vehicles, so we scale the DB updates up to 12 generators.

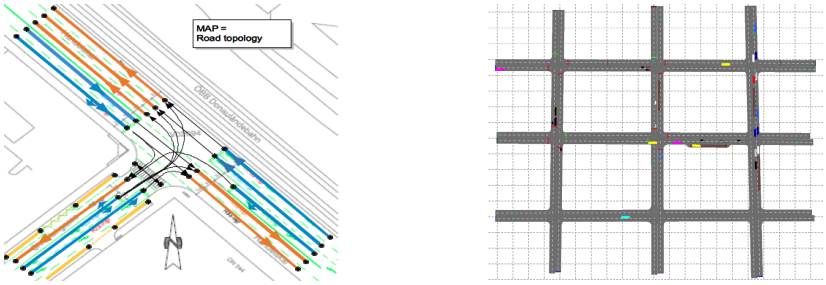

Fig. 2: Schematic representation of scenario (a) and (b)

	Type	#Q	#A	(a) t with #vehicles						(b) t with ms sim. delay			
				10	100	500	1000	2500	5000	0	100	250	500
q_1	O, F	1	1	0.85	0.82	0.91	1.05	1.22	1.58	0.78	0.74	0.73	0.71
q_2	O, F^*	1	6	0.83	0.83	0.83	0.83	0.83	0.83	0.77	0.77	0.72	0.71
q_3	O, S, F	3	23	0.89	0.87	1.00	1.25	1.39	1.74	0.83	0.81	0.77	0.75
q_4	O, S, F	3	22	1.10	1.09	1.24	1.53	1.81	2.32	1.02	1.00	0.95	0.93
q_5	O, S, F	3	42	1.11	1.10	1.26	1.39	1.90	1.92	1.05	1.00	0.98	0.96
q_6	O, S, F	7	52	1.39	1.39	1.49	1.69	2.36	2.28	1.40	1.28	1.26	1.25
q_7	O, F^*	6	69	1.16	1.16	1.16	1.16	1.16	1.16	1.15	1.12	1.11	1.09
q_8	O, S	9	73	0.92	0.94	1.30	1.43	1.72	2.19	0.99	0.98	0.92	0.91
q_9	O, F	9	105	1.67	1.73	1.99	2.06	2.49	2.97	1.71	1.68	1.66	1.63

Table 2: Results (t in secs) for scenario (a) and (b), marked with * are signal streams

this shortens evaluation by roughly 0.2s. Initial evaluation of the queries q_4, q_5, q_6 and q_9 show that with each new stream subquery the number of results dropped down to zero, which seems an implementation issue of PIPELINEDB with *Continuous Views* on the same stream with different window sizes. We found a workaround by adding a delay of 0.2s which again increases the number of results. This delay increases the AET, e.g. by 0.76s in q_9, and might be ignored with future versions of PIPELINEDB and other stream RDBMS. The synthetic queries with mostly ontology (q_6), spatial (q_7), and stream atoms (q_9) clearly show that the challenging part of query evaluation are the stream aggregates. The good performance of PIPELINEDB allows us to work on condensed results (reducing the join sizes); however, stream aggregates could be further accelerated by calculated continuously inline aggregates on the DB, which are skimmed by our queries. Notably, PIPELINEDB keeps not always the order of inserted data points; this does not affect our bag semantics.

In general, our approach is designed to retain *complete* results; however, completeness might be lost as (1) the underlying spatial-stream RDBMS loses results as described above; (2) evaluation of a subquery is slow and subsequent queries start too late. One can solve (1) at the level of the spatial-stream RDBMS, and (2) can be overcome by continuous inline aggregates and query parallelization. In conclusion, the experiments show that the AET of our experimental prototype is for up to 500 vehicles below 1.5s (except q_9). This suggests that with optimizations, e.g. quick detection of red-light violations on complex intersections is feasible.

8 Related Work and Conclusion

Data stream management systems (DSMSs) such as STREAM [3], were built supporting streaming applications by extending RDMBS [14]. More recently, RDF stream processing

engines, such as C-SPARQL [5], SPARQLstream [8], and CQELS [17], were proposed for processing RDF streams integrated with other Linked Data sources. Besides C-SPARQL, most of them follow the DSMSs paradigm and do not support stream reasoning. EP-SPARQL [2] resp. LARS [6] proposes a language that extends SPARQL resp. CQ with stream reasoning, but translates KBs into expressive (less efficient) logic programs. Regarding spatio-temporal RDF stream processing, a few SPARQL extensions were proposed, such as SPARQL-ST [21] and st-SPARQL [16]. Closest to our work are (i) [22], which supports spatial operators as well as aggregate functions over temporal features (ii) [8], which allows evaluating OQA queries over stream RDBMS, and (iii) [20], which extends SPARQL with aggregate functions (using advanced statistics) evaluated over streamed and ordered ABoxes. This work differs regarding (a) the evaluation approach using EAQ with aggregates on the query and not ontology level, (b) hypergraph-based query decomposition, and (c) the main focus of querying streams of spatial data in an OQA setting.

Our approach is situated in-between "classical" stream processing approaches that handle the streaming data as bags in windows, and temporal QA over DL-Lite using temporal operators like LTL in [4], which are evaluated over a (two-sorted) model separating the object and temporal domain. We believe that detemporalization with its bag semantics suffices for the C-ITS case, since the order of V2X messages is not guaranteed, and for most of the normal as well as spatial aggregates it can be ignored (e.g., sum) or is implicit in the data (e.g., Euclidian distance of points). Besides [4], similar temporal QA is investigated in [7] and [15], which are all on the theoretical side and provide no implementation yet. Finally, we build on the results for EAQs in [10], but we introduce spatial streams and more complex queries.

This work is sparked by the LDM for V2X communications, which serves as an integration effort for streaming data (e.g., vehicle movements) in a spatial context (e.g., intersections) over a complex domain (e.g., a mobility ontology). We introduced a suitable approach using ontology-mediated QA for realizing the LDM. For spatial-streaming queries, bridging the gap between stream processing and ontology-mediated QA is not straightforward; we extended previous work in [12] and used epistemic aggregate queries to detemporalize the stream sources. The latter preserves FO-rewritability, which allows us to evaluate conjunctive queries with spatial atoms over existing stream RDBMSs. We also provided a technique to construct query execution plans using hypergraph decomposition, and we have implemented a proof-of-concept prototype to assess the feasibility of our approach on two experiments with mobility data. The results are encouraging, as the evaluation time appeared to be moderate already without optimization.

Future work. Our ongoing and future research is directed to advance the theoretical and practical aspects of our approach. On the theoretical side, a detailed correctness proof for the algorithm that accounts for all different aggregate functions is needed. So far, consistency for QA is neglected and could be enforced in different ways by repairs. The query language could be lifted to SPARQL, but epistemic aggregates, query decomposition, and spatial relations would need reevaluation. On the practical side, our implementation should be extended to pull-based QA with extensive caching and inline aggregates on the DB, along with other optimizations, such as using the pulse for pre-caching resp. window size optimizations, and different query rewriting techniques. Also more complex spatial aggregates, i.e., trajectories, should be considered. Furthermore, cyclic queries need to

be handled. The implementation could be tested in more complex scenarios like event detection (e.g., bus delays) with public transport data.

References

1. Andreone, L., Brignolo, R., Damiani, S., Sommariva, F., Vivo, G., Marco, S.: Safespot final report. Tech. Rep. D8.1.1 (2010), available online.
2. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: Proc. of WWW 2011. pp. 635–644 (2011)
3. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. VLDB J. 15(2), 121–142 (2006)
4. Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F., Zakharyaschev, M.: First-Order Rewritability of Temporal Ontology-Mediated Queries. In: IJCAI'15. 2706–2712 (2015)
5. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: C-sparql: a continuous query language for rdf data streams. Int. J. Semantic Computing 4(1), 3–25 (2010)
6. Beck, H., Dao-Tran, M., Eiter, T., Fink, M.: LARS: A logic-based framework for analyzing reasoning over streams. In: Proc. of AAAI 2015. pp. 1431–1438 (2015)
7. Borgwardt, S., Lippmann, M., Thost, V.: Temporalizing rewritable query languages over knowledge bases. J. Web Sem. 33, 50–70 (2015)
8. Calbimonte, J.P., Mora, J., Corcho, Ó.: Query Rewriting in RDF Stream Processing. In: Proc. of ESWC 2016. pp. 486–502 (2016)
9. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *dl-lite* family. J. Autom. Reasoning 39(3), 385–429 (2007)
10. Calvanese, D., Kharlamov, E., Nutt, W., Thorne, C.: Aggregate queries over ontologies. In: Proc. of ONISW 2008. pp. 97–104 (2008)
11. Eiter, T., Füreder, H., Kasslatter, F., Parreira, J.X., Schneider, P.: Towards a semantically enriched local dynamic map. In: Proc. of 23rd World Congress on Intelligent Transport Systems 2016 (2016)
12. Eiter, T., Krennwallner, T., Schneider, P.: Lightweight spatial conjunctive query answering using keywords. In: Proc. of ESWC 2013. pp. 243–258 (2013)
13. Eiter, T., Parreira, J.X., Schneider, P.: Towards spatial ontology-mediated query answering over mobility streams. In: Proc. of Stream Reasoning Workshop 2016, pp. 13–24 (2016)
14. Golab, L., Özsu, M.T.: Issues in data stream management. SIGMOD Rec. 32(2), 5–14 (2003)
15. Klarman, S., Meyer, T.: Querying temporal databases via OWL 2 QL. In: Proc. of RR 2014. pp. 92–107 (2014)
16. Koubarakis, M., Kyzirakos, K.: Modeling and querying metadata in the semantic sensor web: The model strdf and the query language stsparql. In: Proc. of ESWC 2010. pp. 425–439 (2010)
17. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: ISWC 2011. pp. 370–388 (2011)
18. Maier, D.: The Theory of Relational Databases. Computer Science Press (1983)
19. Netten, B., Kester, L., Wedemeijer, H., Passchier, I., Driessen, B.: Dynamap: A dynamic map for road side its stations. In: Proc. of ITS World Congress 2013 (2013)
20. Özçep, Ö.L., Möller, R., Neuenstadt, C.: Stream-query compilation with ontologies. In: Proc. of AI 2015. pp. 457–463 (2015)
21. Perry, M., Jain, P., Sheth, A.P.: SPARQL-ST: extending SPARQL to support spatiotemporal queries. Geospatial Semantics and the Semantic Web 12, 61–86 (2011)
22. Quoc, H.N.M., Le Phuoc, D.: An elastic and scalable spatiotemporal query processing for linked sensor data. In: Proc. of SEMANTICS 2015. pp. 17–24. ACM (2015)
23. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: Proc. of ISWC 2013. pp. 558–573 (2013)
24. Stocker, M., Smith, M.: Owlgres: A scalable owl reasoner. In: Proc. of OWLED 2008 (2008)