Semantics of Stream Query Languages Using Ontologies*

Özgür L. Özçep, Ralf Möller, and Christian Neuenstadt

Institute for Softwaresystems (STS)
Hamburg University of Technology
Hamburg, Germany
{oezguer.oezcep,moeller,christian.neuenstadt}@tu-harburg.de

Abstract. In order to satisfy the industrial requirements of processing dynamic knowledge from possibly heterogeneous sources, various query languages have been proposed which aim at lifting the relational stream query languages to the ontological/semantic layer. Most of the languages adapt important relational stream operators such as sliding window operators to this layer. But the adaptation and the semantic specification of the stream operators is not trivial due to the additional entailments and constraints w.r.t. ontologies. In this paper, we discuss the semantics of the new query-language framework STARQL (Streaming and Temporal ontology Access with a Reasoning-Based Query Language) in comparison to other stream-temporal query languages and provide results stating its connections to the semantics for temporal logics over ontologies.

Keywords: stream processing, monitoring, ontology, semantics, temporal logics

1 Introduction

In industrial applications, the heterogeneity of the data is a challenging factor for information processing systems that are intended to provide a human user interface—allowing for an equally time-efficient and intuitive access to the data. Ontologies have been considered as an adequate solution to cope with heterogeneity because they, first, provide a common vocabulary (signature) for the data, second, allow for the representation of constraints on the data by axioms, and last, provide a neat semantics. The latter is the theoretical basis on which important services w.r.t. the data such as testing for consistency, deducing new knowledge and, most importantly, querying the data can be realized.

The concrete realizations of the query answering service depends on the requirements of the use case and may follow different ontology-based data access paradigms (OBDA). For OBDA in a wide sense, e.g., ABDEO (Accessing Big Data over Expressive Ontologies) [10], full-fledged reasoners (possibly enhanced

* This work has been supported by the European Commission as part of the FP7 project Optique.

with modularization techniques) are used to provide the service of query answering. OBDA [6] in the strict sense does not use reasoning on the ontology but rewrites the query w.r.t the axioms of the ontology. Mappings are used to define virtual views of the data in the ontology, which are then used to unfold the rewritten query to queries over the backend data sources.

The adoption and successful use of OBDA/ABDEO within the industry depends crucially on the paradigms' means to correctly handle the dynamics of the data, and in particular, to correctly handle temporal and streaming data as used, e.g., in real-time monitoring applications. In this paper, we discuss the semantics of the query framework STARQL (Streaming and Temporal ontology Access with a Reasoning-Based Query Language), which is a new contribution to the recent venture of temporalizing and streamifying OBDA [2,4,7,5,13]. We explicate how the requirements on STARQL and in particular its framework character (calling to handle both OBDA and ABDEO), lead to a specific semantics that strictly separates between the semantics of the plugged-in components and the semantics on top of them. But additionally, we show that for an OBDA instantiation of a STARQL fragment a different, more temporal-logic oriented semantics exists that leads to the same set of answers as w.r.t. the former semantics. We argue that the STARQL fragment is still as useful as linear temporal logic like query languages such as TCQ [4] by embedding TCQ into the STARQL fragment.

The paper is organized as follows: After some preliminary terminology in Sect. 2, we discuss the main idea of processing streams with a window operator (Sect. 3). The syntax and semantics of STARQL is given in Sect. 4. The new semantics and the theorem regarding the equality is stated in Sect. 5. In Sect. 6 before the conclusion we show that TCQ is embeddable into STARQL.

2 Logical Preliminaries

As we are going to deal with query languages referring to ontologies we describe here the necessary terminology from description logics (DLs). An ontology is defined as a triple $\mathcal{O} = \langle Sig, \mathcal{A}, \mathcal{T} \rangle$ with a signature Sig, an ABox $\mathcal{A}$ (set of assertional axioms), and a TBox $\mathcal{T}$ (set of terminological axioms). In all DLs, Sig is made up by subsets of a set of concept symbols N_C, a set of role symbols N_R, and a set of individual constant symbols N_I. DLs with concrete domains or datatypes also allow for additional constants (and predicates) with fixed meanings over the concrete domain. In the following, we drop the signature in the denotation of ontologies. The DLs differ in the set of concept/role constructors they offer and in the constraints for building TBox and ABox axioms.

The STARQL framework allows to plugin different DLs, but here we are focussing on a family of lightweight DLs, the DL-Lite family, which was developed within the OBDA paradigm (in the strict sense) [6]. In strict OBDA, declarative mappings are used to lift data in backend DBs to the ontology level, thereby producing a (virtual) ABox. Query answering in OBDA then follows three main steps: Rewriting the given query w.r.t. the TBox alone; transforming (unfolding)

the rewritten query to queries over the backend DBs w.r.t. the mappings; and last evaluating the unfolded queries over the backend DBs and returning the answers as answers to the original query.

DL-Lite basic concepts B and concepts C are given by the following grammar, where $P \in N_R$, $A \in N_C$ and $a, b \in N_I$: $R \longrightarrow P \mid P^-$; $B \longrightarrow A \mid \exists R$; $C \longrightarrow B \mid \neg B$. ABox axioms have the form: $A(a), R(a, b)$. TBox axioms have the form $B \sqsubseteq C$, (func R), or $R_1 \sqsubseteq R_2$, where Rs that are declared functional are not allowed to occur on the right. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, (\cdot)^{\mathcal{I}})$ consist of a domain $\Delta^{\mathcal{I}}$ and a denotation function $(\cdot)^{\mathcal{I}}$ that maps constants a to elements $(a)^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, atomic concepts A to sets $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and roles P to binary relations $(P)^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. For complex concepts we have: $(P^-)^{\mathcal{I}} = \{(d, e) \mid (e, d) \in P^{\mathcal{I}}\}$; $(\exists R)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists e.(d, e) \in R^{\mathcal{I}}\}$; $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$. The satisfaction relation between interpretations and axioms is given by: $\mathcal{I} \models B \sqsubseteq C$ iff $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$; $\mathcal{I} \models R_1 \sqsubseteq R_2$ iff $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$; $\mathcal{I} \models B(a)$ iff $a^{\mathcal{I}} \in B^{\mathcal{I}}$; $\mathcal{I} \models R(a, b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$; $\mathcal{I} \models$ (func R) iff $R^{\mathcal{I}}$ is a (partial) function.

In the whole paper, let $[n] = \{0, 1, \ldots, n\}$ be the natural numbers up to $n \in \mathbb{N}$. A *conjunctive query* $\exists y_1, \ldots, y_m.\psi(y_1, \ldots, y_m, x_1, \ldots, x_n)$ consists of a prefix of existential quantifiers and is a conjunction ψ of atoms of the form $A(z)$ or $R(z, w)$ with $z, w \in \{y_1, \ldots, y_m, x_1, \ldots, x_n\}$ or $z, w \in N_I$. The y_i are bound by the existential quantifiers, the free variables x_i are the output slots of the query. If the set of free variables is empty, the query is called Boolean. This is the logical notation from description logics. In a more SPARQL like syntax a CQ of the form $\exists y A(x) \wedge R(x, y)$ would be written as `SELECT ?x WHERE { ?x :a A AND ?y :R ?y }`. We are going to switch freely between this notational variants. We use vector notation for lists of variables, e.g., $\boldsymbol{x} = x_1, \ldots, x_n$, and so $\exists \boldsymbol{x}$ is an abbreviation for $\exists x_1, \ldots, \exists x_n$.

A union of CQs ψ, termed UCQ, is a disjunction of CQs ϕ_i with the same set of free variables $\psi(\boldsymbol{x}) = \exists \boldsymbol{y_1} \phi_1(\boldsymbol{x}, \boldsymbol{y_1}) \vee \exists \boldsymbol{y_2} \phi_2(\boldsymbol{x}, \boldsymbol{y_2}) \vee \cdots \vee \exists \boldsymbol{y_n} \phi_n(\boldsymbol{x}, \boldsymbol{y_n})$. Generally, when writing $\phi(\boldsymbol{x})$ we mean that $\boldsymbol{x}$ are the free variables in ϕ. For a vector of constants $\boldsymbol{a}$ of length n and a formula ϕ with free variables $\boldsymbol{x}$ of the same length n, $\phi(a)$ is the Boolean query resulting from substituting a_i for x_i, $i \in [n]$. For a query $\phi(\boldsymbol{x})$ (not necessarily a UCQ) and an interpretation $\mathcal{I}$, the set of answers in $\mathcal{I}$ is $ans(\phi(\boldsymbol{x}), \mathcal{I}) = \{\boldsymbol{a} = (a_1, \ldots, a_n) \mid a_i \in N_I \text{ and } \mathcal{I} \models \phi(\boldsymbol{a})\}$. The set of *certain answers* w.r.t. an ontology $\langle \mathcal{A}, \mathcal{T} \rangle$ is defined as the intersection of the answers of ϕ on all models of $\langle \mathcal{A}, \mathcal{T} \rangle$: $cert(\phi(\boldsymbol{x}), \langle \mathcal{A}, \mathcal{T} \rangle) = \bigcap_{\mathcal{I} \models \langle \mathcal{A}, \mathcal{T} \rangle} ans(\phi, \mathcal{I})$. For inconsistent ontologies $\boldsymbol{x}$ is bound to all possible bindings $\boldsymbol{a}$ over all constants appearing in the ontology.

The notion of (perfect) rewritability is explicated as follows: For every ABox $\mathcal{A}$ let $DB(\mathcal{A})$ be its minimal herbrand model. Let QL_1 and QL_2 be query languages over the same signature of an ontology and OL be an ontology language. QL_1 allows for QL_2-rewriting of query answering w.r.t. the ontology language OL iff for all queries ϕ in QL_1 and TBoxes $\mathcal{T}$ in OL there exists a query $\phi_{\mathcal{T}}$ in QL_2 such that for all ABoxes $\mathcal{A}$ it holds that: $cert(\phi, \langle \mathcal{A}, \mathcal{T} \rangle) = ans(\phi_{\mathcal{T}}, DB(\mathcal{A}))$. A particularly interesting case is QL_2 = first-order logic (FOL) queries. A well-mown fact [6] is: UCQs are FOL rewritable w.r.t. DL-Lite ontologies.

Next to rewriting, unfolding has also constraints on the used languages in strict OBDA. The backend DBs are in most cases relational DBs or relational data stream management systems (DSMS) which are equipped with SQL like query languages. As these fulfill the property of domain independence, the unfolding must give a domain independent query. Domain independence of a query language means that the answer of a query does not depend on the domain but only on the constants occurring in a DB. Here is the formal definition [3]: A query ϕ is *domain independent* iff for all interpretations $\mathcal{I}, \mathcal{J}$ such that $\mathcal{I}$ is a substructure of $\mathcal{J}$: $ans(\phi, \mathcal{I}) = ans(\phi, \mathcal{J})$.

3 Semantics for Stream Query Languages

A *stream* is a (possibly infinite) sequence of elements from some domain D. Though streams are considered as sequences, we are going to use set notation also for the sequence. The focus of this paper is on *temporal streams*, which consist of elements from some domain D together with a timestamp from some temporal domain. In this paper, we do not deal with out-of-order arrivals of elements: i.e., the timestamps of elements are assumed to be a monotonically increasing w.r.t. the sequence ordering.

The domain D in case of relational stream query languages are tuples (adhering to some relational schema). In the semantic scenario, which is the focus of this paper, different types of domains D are considered. First of all, the inputs of the queries are streams of timestamped ABox axioms (timestamped RDF triples). Moreover, there may be streams of bindings (tuples of constants not necessarily adhering to a schema) and in the case of STARQL we are going to consider also streams of whole (temporal) ABoxes (RDF graphs).

The temporal domain is fixed by a structure $(T, \leqslant)$ with a set T of time points (to be used as timestamps) and a binary relation $\leqslant$ on it. The set may be discrete ($T = \mathbb{N}$), or dense ($T = \mathbb{Q}$) or even continuous ($T = \mathbb{R}$). Working with the latter domains is possible due to the fact that streams are defined as sequences. The relation $\leqslant$ is assumed to be a linear order (reflexive, transitive, asymmetrical, connected) on T.

Stream query languages for data stream management systems (DSMS) have to cope with the potential infinity of streams, which makes it impossible to apply the usual SQL blocking operators. The solution is to provide a window operator the content of which is a dynamically updated finite part of the stream. Classical window operators such as that defined in the relational stream query language CQL [1] have a fixed window width (also called range) and a sliding parameter, which determines the update frequency.

In CQL, the window operator produces for every time $t \in T$ a set of tuples whose timestamps t' are in the interval $[t - r, t]$ for the range parameter r. The original timestamps t' (if not stored in an extra attribute) get lost in the window contents. Forgetfulness may not be problematic for relational streams (but even here one has to deal with key declarations), but on streams of semantically constrained assertions, the window operator has to be handled with care. For

example, if the ontology has a constraint of the form (func val) saying that at every time point a sensor may have at most one value, the naive application of the window operator on measurement streams may directly lead to inconsistency: If sensor $s0$ shows value 90° at time $t = 3s$ and 95° at time $t = 4s$, then the application of a window operator with range at least 2s leads to the inconsistent ABox $\{val(s0, 90), val(s0, 95)\}$. The need to deal with potential inconsistencies in stream scenarios has also been pointed out in [14].

All of the recent stream query languages on the ontological level, e.g., [7,5,13], use a window operator as that of CQL. There are different reasons why they do not run into the inconsistency problem mentioned above: Either the TBox language is not expressive enough so that the ontology is always consistent; or the window operator's contents is defined such that it does not generate sets of ABox axioms (sets of RDF triples) but only bindings not governed by any constraint; or last but not least, the approach uses reification, where one does not say, e.g., that the assertion $val(s0, 90)$ holds at time point $3s$, but instead that there is a measurement object, having an associated sensor s and value 90 and a time $3s$.

All of the three "solutions" mentioned above could not be followed with the STARQL framework: STARQL allows the use of expressive DLs in order to cope with the demands of industrial applications for expressing non-trivial diagnostics-symptoms relations; it produces streams of ABox assertions in order to provide an orthogonal query language; and it uses a non-reified approach (for different reasons we refer the reader to the discussion in [8]). Hence STARQL has to provide a different window semantics. This is going to be explicated in the following section.

4 The STARQL Framework

One of the most important scenarios for stream processing are real-time monitoring applications where streams of sensor measurements and event data are processed. Data on the infrastructure of the sensors and the underlying assemblies or components, the sensors are attached at, are normally stored in an heterogeneous set of flat data files, which typically do not adhere to a specific schema, contain different units, are incomplete etc. Using ontologies as a homogeneous interface to the infrastructure and stream data can simplify (in terms of conceptual demands to the user as wells as the time needed) the access, retrieval, and manipulation of the data.

In this example, we consider the case where many gas turbines have many thousands of sensors and control units attached on different parts of them, sending with high frequency measurement data (such as rotation speed, temperature, pressure) and event messages (failures, status messages etc.) A typical request relying on real requirements of engineers is the following information need: Mark every 10 seconds all temperature sensors as critical (together with the time point) such that the following holds: In the last 5 minutes there has been some monotonically increase on an interval of length at least 2 minutes followed by a failure

message of category A. A STARQL formalization of this information need is given in the following listing.

```
1  CREATE STREAM Sout AS
2  CREATE PULSE AS START = 0s, FREQUENCE = 10s
3  CONSTRUCT   { ?s :a inCriticalState } <NOW>
4  FROM SMsmt[NOW-5min, NOW]->10s, <http://ABox>, <http://TBox>
5  WHERE { ?s :a TempSens }
6  SEQUENCE BY StdSeq AS SEQ
7  HAVING
8  EXISTS i1, i2, i3 in SEQ
9   0 < i1 AND i2 < max AND i3 = i2 + 1 AND
10  ts(i2) - ts(i1) >= 2min AND
11  GRAPH i3 { ?s :message ?m  . ?m :a A-Message } AND
12  FORALL i, j in SEQ,?x,?y:
13   IF  i1 <= i  AND i<= j  AND j <= i2  AND
14       GRAPH i { ?s :val ?x }  AND GRAPH j { ?s :val ?y }
15   THEN ?x <= ?y
```

After the create declarations, the `CONSTRUCT` operator is used to fixe the output format for the output stream, giving for every time point `NOW` the critical sensors in RDF quadruple notation. The window operator `[NOW-5min, NOW]-> 10s` gives snapshots of the stream with the specified update frequency of 10s and range of 5 minutes. The `WHERE` clause (l. 5) determines the temperature sensors $?s$. These may not be explicitly stated in the static ABox but only deducible w.r.t. the TBox, which contains axioms of the form $BurnerTipTempSens \sqsubseteq TempSens$. For every binding for $?s$, the query tests in the `HAVING` clause (l. 7–15) the specified conditions. STARQL uses a sequence operator to build sequences of states that can be referred to with variables i, j. In combination with standard sequencing `StdSeq`, the original timestamps can be referred to with a function `ts()`. In order to ask whether a condition holds at state i, the subgraph notation from RDF/SPARQL is used. So, e.g., the state atom (in line 13) `GRAPH i3 {?s :message ?m . ?m :a A-Message}` asks whether `?s` showed a message of Type A at state `i3`. Here, `i3` is specified as the successor state of end state `i2` in the interval `[i1, i2]` for which one tests monotonicity (`FORALL` condition, l. 12–15). Below we give the syntax and semantics in more detail.

4.1 Syntax

Figures 1 and 2 contain the main structure for a simplified fragment of the STARQL query framework. The simplifications are: We don't deal with aggregation operators, with the slack operator, concrete domains, and negation. (For the full definitions see [12] and [11], which uses safety conditions. But note, e.g., that in order to use an `FORALL` quantifier we have to "filter" the variables $\boldsymbol{x}$ with a having clause, thereby making $\boldsymbol{x}$ safe.) The grammar STARQL(OL,ECL) contains parameters that have to be specified in its instantiations. Namely, one has to specify the ontology language OL and the embedded condition language

$$
\begin{aligned}
\mathit{createExp} &\longrightarrow \texttt{CREATE STREAM } \mathit{sName} \texttt{ AS } [\mathit{pulseExp}]\ \mathit{constrExp}\\
\mathit{pulseExp} &\longrightarrow \texttt{CREATE PULSE AS START = } \mathit{startTime}, \texttt{ FREQUENCE = } \mathit{freq}\\
\mathit{constrExp} &\longrightarrow \texttt{CONSTRUCT } \mathit{constrHead}(\boldsymbol{x},\boldsymbol{y})\\
&\quad\ \texttt{FROM } \mathit{listWinStreamExp} [\texttt{ , } \underline{\mathit{URISToABoxesAndTBoxes}}]\\
&\quad\ \texttt{WHERE } \mathit{whereClause}(\boldsymbol{x})\\
&\quad\ \texttt{SEQUENCE BY } \mathit{seqMethod}\\
&\quad\ \texttt{HAVING } \mathit{safeHavingClause}(\boldsymbol{x},\boldsymbol{y})\\
\mathit{constrHead}(\boldsymbol{x},\boldsymbol{y}) &\longrightarrow \mathit{TriplePat}(\boldsymbol{x},\boldsymbol{y})\ \texttt{<}\mathit{timeExp}\texttt{>}\ \{\ \texttt{.}\ \mathit{constrHead}(\boldsymbol{x},\boldsymbol{y})\}\\
\mathit{listWinStreamExp} &\longrightarrow (\mathit{sName} \mid \mathit{constrExp})\mathit{windowExp}[\texttt{ , }\ \mathit{listWinStreamExp}]\\
\mathit{windowExp} &\longrightarrow \texttt{[}\mathit{timeExp}_1, \mathit{timeExp}_2\texttt{]->}\mathit{sl}\\
\mathit{whereClause}(\boldsymbol{x}) &\longrightarrow \mathit{ECL}(\boldsymbol{x})\\
\mathit{seqMethod} &\longrightarrow \mathit{StdSeq} \mid \mathit{SeqMethod}(\sim)
\end{aligned}
$$

Fig. 1: Syntax for STARQL(OL, ECL) template without `HAVING` clauses

ECL. ECL is a query language referring to the signature of the ontology language. STARQL uses ECL conditions as atoms in its `WHERE` and `HAVING` clauses. The adequate instantiation of STARQL(OL, ECL) may vary depending on the requirements of the use case. We mention here that in some scenarios a very expressive description logic such as $\mathcal{SHI}$ combined with the query language of grounded conjunctive queries (GCQs), as used in [10] in the ABDEO paradigm may be in order. In Sect. 5 we are going to work with the specific OBDA instantiation STARQL(DL-Lite, UCQ) (see the preliminaries for the definitions).

4.2 Semantics

In order to define the semantics for instantiations of STARQL(OL,ECL), parameter values for OL and ECL have to fulfill the following conditions: There must be notion of certain answers of an ECL w.r.t. an ontology. We recapitulate the original semantics of STARQL queries as defined in [11] in this section.

Assume that we have the following STARQL query, the denotation $[\![\cdot]\!]$ of which is going be fixed by denotations of its components.

$$
\begin{aligned}
&\texttt{CONSTRUCT } \Theta_1(\boldsymbol{x},\boldsymbol{y})\langle \mathit{timeExp}_1\rangle, \ldots, \Theta_r(\boldsymbol{x},\boldsymbol{y})\langle \mathit{timeExp}_r\rangle\\
&\texttt{FROM } S_1\ \mathit{winExp}_1, \ldots, S_m\ \mathit{winExp}_m, \mathcal{A}^0_{st}, \ldots, \mathcal{A}^k_{st}, \mathcal{T}^0, \ldots, \mathcal{T}^l\\
&\texttt{WHERE } \psi(\boldsymbol{x})\ \texttt{SEQUENCE BY } \mathit{seqMeth}\ \texttt{HAVING } \phi(\boldsymbol{x},\boldsymbol{y})
\end{aligned}
$$

The input of STARQL queries are static ABoxes, TBoxes, and streams of timestamped ABox assertions (timestamped RDF triples, or RDF quadruples according to [9]). The output is a new stream of timestamped ABox assertions. STARQL, as of now, considers only non-temporal TBoxes: By this we mean that there are no special temporal or stream constructors; there may be a time attribute but this gets no special treatment as a logical operator.

$$
\begin{aligned}
term(i) &\longrightarrow i \\
term() &\longrightarrow \texttt{max} \mid \texttt{0} \mid \texttt{1} \\
arithAtom(i_1, i_2) &\longrightarrow term_1(i_1)\ op\ term_2(i_2) \quad (op \in \{\texttt{<}, \texttt{<=}, \texttt{=}, \texttt{>}, \texttt{>=}\}) \\
arithAtom(i_1, i_2, i_3) &\longrightarrow \texttt{plus}(term_1(i_1), term_2(i_2), term_3(i_3)) \\
stateAtom(\boldsymbol{x}, i) &\longrightarrow \texttt{GRAPH i}\ \ ECL\ (\boldsymbol{x}) \\
atom(\boldsymbol{x}) &\longrightarrow arithAtom(\boldsymbol{x}) \mid stateAtom(\boldsymbol{x}) \\
hCl(\boldsymbol{x}) &\longrightarrow atom(x) \mid hCl(\boldsymbol{x})\ \texttt{OR}\ hCl(\boldsymbol{x}) \\
hCl(\boldsymbol{x}, \boldsymbol{y}) &\longrightarrow hCl(\boldsymbol{x})\ \texttt{AND}\ hCl(\boldsymbol{y}) \\
hCl(\boldsymbol{x}) &\longrightarrow hCl(\boldsymbol{x})\ \texttt{AND FORALL}\ \boldsymbol{y}\ \texttt{IF}\ hCl(\boldsymbol{x}, \boldsymbol{y})\ \texttt{THEN}\ hCl(\boldsymbol{x}, \boldsymbol{y}) \\
hCl(\boldsymbol{x}, \boldsymbol{z}) &\longrightarrow \texttt{EXISTS}\ \boldsymbol{y}\ hCl(\boldsymbol{x}, \boldsymbol{y})\ \texttt{AND}\ hCl(\boldsymbol{z}, \boldsymbol{y}) \\
safeHavingClause(\boldsymbol{x}) &\longrightarrow hCl(\boldsymbol{x}) \quad \text{(where } \boldsymbol{x} \text{ does not contain an } \texttt{i} \text{ variable)}
\end{aligned}
$$

Fig. 2: HAVING grammar template for STARQL(OL, ECL)

WHERE Clause Let $\mathcal{A}_{st} = \bigcup_{i\in[k]} \mathcal{A}^i_{st}$ and $\mathcal{T} = \bigcup_{i\in[l]} \mathcal{T}^i$. In the `WHERE` clause only $\mathcal{A}_{st}$ and $\mathcal{T}$ are relevant for the answers. So, purely static conditions (e.g. asking for sensor types as in the example above) are evaluated only on the static ABoxes. The result are bindings $\boldsymbol{a}_{wh} \in cert(\phi(\boldsymbol{x}), \langle \mathcal{A}_{st}, \mathcal{T} \rangle)$. This set of bindings is applied to the `HAVING` clause $\phi(\boldsymbol{x}, \boldsymbol{y})$ resulting in the clause $\phi(\boldsymbol{a_{wh}}, \boldsymbol{y})$.

Window Operator. Let $[\![S_i]\!]$ for $i \in [m]$ denote the input streams. We define the denotation of the windowed stream $ws_i = S_i\ \texttt{[}timeExp^i_1, timeExp^i_2\texttt{]->}sl_i$. To keep readability we drop the index $i \in [m]$ and just talk of the stream S and the windowed stream ws.

The denotation of ws is a stream with timestamps from the set $T' \subseteq T$, where T' is fixed by the pulse declaration. The elements within the stream are temporal ABoxes. Under a *temporal ABox* we understand a set of timestamped ABox axioms. This is the crucial point where the window semantics in STARQL differs from that in [1,7,5,13]: Within the window content the original timestamps of the ABox axioms from the input streams are preserved.

Assume that $\lambda t.g_1(t) = [\![timeExp_1]\!]$ and $\lambda t.g_2(t) = [\![timeExp_2]\!]$ are the unary functions of time denoted by the time expressions in the window. Let T' be represented by the increasing sequence of timestamps $(t_i)_{i\in\mathbb{N}}$, where t_0 is the starting point fixed in the pulse declaration. We have to define for every t_i the temporal ABox $\tilde{\mathcal{A}}_{t_i}\langle t_i \rangle \in [\![wS]\!]$. If $t_i < sl - 1$, then $\tilde{\mathcal{A}}_{t_i} = \varnothing$. Else set first $t_{start} = \lfloor t_i/sl \rfloor \times sl$ and $t_{end} = max\{t_{start} - (g_2(t_i) - g_1(t_i)), 0\}$, and define on that basis $\tilde{\mathcal{A}}_{t_i} = \{ax\langle t \rangle \mid ax\langle t \rangle \in [\![S]\!] \text{ and } t_{end} \leqslant t \leqslant t_{start}\}$. Now, the denotations of all windowed streams, which are streams of temporal ABoxes are are joined w.r.t. the timestamps in T':

$$jstr = \{(\bigcup_{j\in[m]} \tilde{\mathcal{A}}^j_t)\langle t \rangle \mid t \in T' \text{ and } \tilde{\mathcal{A}}^j_t\langle t \rangle \in [\![ws_j]\!]\}$$

As one can see, the pulse declaration serves the purpose of syncing up the local sliding parameters in the streams S_i.

Sequencing. The joined stream $jstr$ is processed according to the sequencing method specified in the STARQL query. The output is a stream with timestamps from T'. The stream elements are finite sequences of pure ABoxes specified in the language OL. We call the structure $\langle (\mathcal{A}_i)_{i\in[n]}, \mathcal{T}\rangle$ consisting of a finite sequence of non-temporal ABoxes and a pure TBox a *sequenced ontology (SO)*.

The sequencing methods used in STARQL refer to an equivalence relation $\sim$ to specify which assertions go into the same ABox. The equivalence classes $[x]_\sim$ for $x \in T$ form a partition of T. We restrict the class of admissible equivalence relations to those $\sim$ that respect the time ordering $\leqslant$, i.e., $\sim$ should be a congruence w.r.t. $\leqslant$: If $t_1 \sim t_2$ and $t_1' \leqslant t_1$, then also $t_1' \leqslant t_2$ (and if $t_1 \leqslant t_1'$, then also $t_2 \leqslant t_1'$). This guarantees that one can also define an ordering $\leqslant$ on the equivalence classes which is presupposed in the `HAVING` clause. The equivalence classes are referred to as states and are denoted by variables i, j etc.

Now, we define the sequence of ABoxes generated by $seqMethod(\sim)$ on the stream of temporal ABoxes as follows: Let $\tilde{\mathcal{A}}_t\langle t\rangle$ be the temporal ABox of $jstr$ at t. Let $T'' = \{t_1, \ldots, t_l\}$ be the time points occurring in $\tilde{\mathcal{A}}_t$ and let k' the number of equivalence classes generated by the time points in T''. Then define the sequence at t as $(\mathcal{A}_0, \ldots, \mathcal{A}_{k'})$ where for every $i \in [k']$ the pure ABox $\mathcal{A}_i$ is $\mathcal{A}_i = \{ax\langle t'\rangle \mid ax\langle t'\rangle \in \tilde{\mathcal{A}}_t \text{ and } t' \text{ in } i^{th} \text{ equivalence class}\}$. The standard sequencing method `StdSeq` is just $seqMethod(=)$. In this case, the resulting states consist of just one time point.

HAVING Clause. STARQL's semantics for the `HAVING` clauses relies on the certain answer semantics of the embedded ECL conditions. We have to define the semantics of $\phi(\boldsymbol{a}, \boldsymbol{y})$ for every binding $\boldsymbol{a}_{wh}$ from the evaluation of the `WHERE` clause. We declare for every t how to get bindings for $\boldsymbol{y}$. Assume that the sequence of ABoxes at t is $seq = (\mathcal{A}_0, \ldots, \mathcal{A}_k)$. The set of bindings to be defined makes up what we call the *separation-based* certain answers, denoted $cert_{sep}$:

$$cert_{sep}(\phi(\boldsymbol{a}_{wh}, \boldsymbol{y}), \langle \mathcal{A}_i \cup \mathcal{A}_{st}, \mathcal{T}\rangle)$$

We distinguish two cases: If for any i the pure ontology $\langle \mathcal{A}_i \cup \mathcal{A}_{st}, \mathcal{T}\rangle$ is inconsistent, then we set $cert_{sep}$ = NIL, where NIL is a new constant not contained in the signature.

In the other case, the bindings are defined as follows. For t one constructs a sorted first order logic structure $\mathcal{I}_t$: The domain of $\mathcal{I}_t$ consists of the index set $\{0, \ldots, k\}$ as well as the set of all individual constants of the signature. For every *stateAtom* `GRAPH i` $ECL(\boldsymbol{z})$ in $\phi(\boldsymbol{a}_{wh}, \boldsymbol{y})$ with free variables $\boldsymbol{z}$ having length l, say, introduce an $(l+1)$-ary symbol R and replace `GRAPH i` $ECL(\boldsymbol{z})$ by $R(\boldsymbol{z}, i)$. The denotation of R in $\mathcal{I}_t$ is then just stipulated as the set of certain answers of the embedded condition $ECL(\boldsymbol{z})$ w.r.t. the i^{th} ABox $\mathcal{A}_i$:

$$R^{\mathcal{I}_t} = \{(\boldsymbol{b}, i) \mid \boldsymbol{b} \in cert(ECL(\boldsymbol{z}), \langle \mathcal{A}_i \cup \mathcal{A}_{st}, \mathcal{T}\rangle)\}$$

Note that also the static ABoxes are integrated to the evaluation: The idea is that static ABoxes must hold at every state.

Constants are denoted by themselves in $\mathcal{I}_t$. This already fixes a structure $\mathcal{I}_t$ with finite denotations of its relation symbols. The evaluation of the `HAVING` clause is then nothing more than evaluating the FOL formula (after the substitutions) on the structure $\mathcal{I}_t$.

Properties. The motivation for such a definition of the semantics of `HAVING` clauses is a strict separation of the semantics provided by the embedded condition languages ECL and the semantics used on top of it. This allows for embedding any ECL without repeatedly redefining the semantics.

The separation-based semantics has an immediate consequence for the property of perfect rewritability, which is adapted to the sequenced setting as follows. Let $\mathcal{O} = \langle (\mathcal{A}_i)_{i\in[n]}, \mathcal{T} \rangle$ be a SO. The sequence of ABoxes has a canonical model $DB((\mathcal{A}_i)_{i\in[n]})$ defined as the sequence of minimal herbrand models $DB(\mathcal{A}_i)$ of the component ABoxes $\mathcal{A}_i$ in the sequence. Let QL_1 and QL_2 be two query languages over the same signature of a SO and OL be a language for the sequenced ontologies SO.

Definition 1. *QL_1 allows for QL_2-rewriting of query answering w.r.t. the ontology language OL iff for all queries ϕ in QL_1 and TBoxes $\mathcal{T}$ in OL there exists a query $\phi_\mathcal{T}$ in QL_2 such that for all $n \in \mathbb{N}$ and all sequences of ABoxes $(\mathcal{A}_i)_{i\in[n]}$ it holds that: $cert(\phi, \langle (\mathcal{A}_i)_{i\in[n]}, \mathcal{T} \rangle) = ans(\phi_\mathcal{T}, DB((\mathcal{A}_i)_{i\in[n]}))$*

Assume that the ECL language w.r.t. OL allows for perfect rewriting such that the rewritten formula is again an ECL condition. We call such an ECL language *rewritability closed* w.r.t. OL. Then, an immediate consequence of the separated semantics is the following observation.

Observation 1 *Let ECL be a rewritability closed condition language and consider the instantiation of `HAVING` called QL_1. Then QL_1 allows for QL_1 rewriting for separation-based certain query answering w.r.t. OL.*

5 Separation-Based versus Holistic Semantics

One may argue that the original semantics of the STARQL `HAVING` clause is not as clear as an integrated/holistic semantics, which would have to define the notion of certain answers for the whole `HAVING` clause. Because of this, we are going to define in this section a holistic semantics for STARQL `HAVING` clauses. Then we show that for a fragment of the `HAVING` clauses we get the same answers. This fragment is shown to capture LTL (linear temporal logic) like query languages and may hence considered to be at least as useful as LTL like languages.

We assume that we have a `HAVING` clause where the free variables $\boldsymbol{x}$ of the `WHERE` clause are already bound. At every evolving time point t we have a microcosm made up by a SO $\langle (\mathcal{A}_i)_{i\in[n_t]}, \mathcal{T} \rangle$. We assume that the ABoxes $\mathcal{A}_i$ already

contain the static ABox $\mathcal{A}_{st}$. The length of the sequence n_t may depend on the time point t. As we now fix it, we write just n for n_t.

Definition 2. *Let $\mathcal{O} = \langle (\mathcal{A}_i)_{i\in[n_t]}, \mathcal{T}\rangle$ be a SO. Let $\hat{\mathcal{I}} = (\mathcal{I}_i)_{0\leqslant i\leqslant n}$ be a sequence of interpretations $\mathcal{I}_i = (\Delta, \cdot^{\mathcal{I}_i})$ over a fixed non-empty domain Δ where the constants' interpretation does not change for different $\mathcal{I}_i, \mathcal{I}_j$. Then $\hat{\mathcal{I}}$ is a model of $\mathcal{O}$ (written $\hat{\mathcal{I}} \models \mathcal{O}$) if $\mathcal{I}_i \models \langle \mathcal{A}_i, \mathcal{T}\rangle$ for all $i \in [n]$.*

All interpretations $\mathcal{I}_i$ have the same domain Δ. The constants' denotations does not change from stet to state, hence they are considered rigid. The TBox $\mathcal{T}$ is assumed to be a non-temporal TBox. Its inclusions hold at every time point.

We see that the set of models of $\mathcal{O} = \langle (A_i)_{i\in[n]}, \mathcal{T}\rangle$ is just the cartesian product of all models $\mathcal{I}_i$ of the local ontologies $\mathcal{T} \cup \mathcal{A}_i$.

Observation 2 $\quad \{\hat{\mathcal{I}} \mid \hat{\mathcal{I}} \models \langle (A_i)_{i\in[n]}, \mathcal{T}\rangle\} = \times_{i\in[n]} \{\mathcal{I} \mid \mathcal{I} \models \langle A_i, \mathcal{T}\rangle\}$

We are going to define a satisfaction relation between a sequence of interpretations and any set of assignments σ to variables on the one hand and Boolean `HAVING` clauses on the other hand.

Definition 3. *Let $\hat{\mathcal{I}} = (\mathcal{I}_i)_{i\in[n]}$ be a sequence of interpretations of length n and σ be an assignment of individuals $d \in \Delta$ to individual variables and numbers $i \in [n]$ to state variables i, j. As usual, define $\sigma[j \mapsto \underline{j}]$ to be that assignment where j is assigned $\underline{j}$.*

$$
\begin{aligned}
\hat{\mathcal{I}}, \sigma &\models \texttt{GRAPH i } \psi \;\; \textit{iff} \;\; \mathcal{I}_{\sigma(i)} \models \psi \\
\hat{\mathcal{I}}, \sigma &\models \texttt{EXISTS i } \phi \;\; \textit{iff} \;\; \textit{there is } \underline{i} \in [n] \textit{ s.t. } \hat{\mathcal{I}}, \sigma[i \mapsto \underline{i}] \models \phi \\
\hat{\mathcal{I}}, \sigma &\models \texttt{FORALL i } \phi \;\; \textit{iff} \;\; \textit{for all } \underline{i} \in [n] \textit{ it holds that } \hat{\mathcal{I}}, \sigma[i \mapsto \underline{i}] \models \phi \\
\hat{\mathcal{I}}, \sigma &\models \texttt{EXISTS x } \phi \;\; \textit{iff} \;\; \textit{there is } d \in \Delta \textit{ s.t. } \hat{\mathcal{I}}, \sigma[x \mapsto d] \models \phi \\
\hat{\mathcal{I}}, \sigma &\models \texttt{FORALL x } \phi \;\; \textit{iff} \;\; \textit{for all } d \in \Delta \textit{ it holds that } \hat{\mathcal{I}}, \sigma[x \mapsto d] \models \phi \\
\hat{\mathcal{I}}, \sigma &\models \phi_1 \texttt{ AND } \phi_2 \;\; \textit{iff} \;\; \hat{\mathcal{I}}, \sigma \models \phi_1 \textit{ and } \hat{\mathcal{I}}, \sigma \models \phi_2 \\
\hat{\mathcal{I}}, \sigma &\models \phi_1 \texttt{ OR } \phi_2 \;\; \textit{iff} \;\; \hat{\mathcal{I}}, \sigma \models \phi_1 \textit{ or } \hat{\mathcal{I}}, \sigma \models \phi_2 \\
\hat{\mathcal{I}}, \sigma &\models \phi_{arithAtom} \;\; \textit{iff} \;\; \mathcal{I}_0, \sigma \models \phi_{arithAtom} \\
\hat{\mathcal{I}} &\models \phi \;\; \textit{iff} \;\; \hat{\mathcal{I}}, \varnothing \models \phi
\end{aligned}
$$

For a `HAVING` clause $\phi(\boldsymbol{x})$ and a sequence of interpretations $\hat{\mathcal{I}}$, the set of answers is $ans(\phi(\boldsymbol{x}), \hat{\mathcal{I}}) = \{\boldsymbol{a} \mid \hat{\mathcal{I}} \models \phi(\boldsymbol{a})\}$. The set of certain answers w.r.t. a SO $\mathcal{O} = \langle (A_i)_{i\in[n]}, \mathcal{T}\rangle$ is: $cert_h(\phi, \mathcal{O}) = \bigcap_{\hat{\mathcal{I}}\models\mathcal{O}} ans(\phi, \hat{\mathcal{I}})$

Now we consider a fragment of the `HAVING` clauses for the instantiation of OL = DL-Lite and ECL = UCQ and denote it by $\mathcal{L}^{\exists}_{HCL}$: We disallow the operators `FORALL x` and `EXISTS x`. The implicit existential quantifiers in the UCQs are kept. We are going to show that the original separation-based semantics and the holistic certain answer semantics (denoted $cert_h$) are the same on this fragment.

Theorem 1. *For any SO $\mathcal{O} = \langle (A_i)_{i\in[n]}, \mathcal{T}\rangle$ and any $\phi \in \mathcal{L}^{\exists}_{HCL}$ the following equality holds: $cert_h(\phi, \mathcal{O}) = cert_{sep}(\phi, \mathcal{O})$.*

In preparing the proof, we transform, for every $n \in \mathbb{N}$, the formulas in $\mathcal{L}^{\exists}_{HCL}$ into formulas (depending on n) that do not contain any quantifiers over the states of the sequence but may contain constants ι_j (for $j \in [n]$) denoting the states. Denote the rewriting by $\tau^n_{\iota_j}$ with parameters ι_j, n.

$\tau^n_{\iota_j}(\texttt{GRAPH i } \psi) = \texttt{GRAPH j } \psi$; $\quad \tau^n_{\iota_j}(\phi_1 \texttt{ AND } \phi_2) = \tau^n_{\iota_j}(\phi_1) \texttt{ AND } \tau^n_{\iota_j}(\phi_2)$
$\tau^n_{\iota_j}(\phi_1 \texttt{ OR } \phi_2) = \tau^n_{\iota_j}(\phi_1) \texttt{ OR } \tau^n_{\iota_j}(\phi_2)$; $\quad \tau^n_{\iota_j}(\phi_{arithAtom}) = \phi_{arithAtom}$
$\tau^n_{\iota_j}(\texttt{EXISTS i } \phi) = \tau^n_0(\phi) \texttt{ OR } \tau^n_1(\phi) \texttt{ OR } \ldots \texttt{ OR } \tau^n_n(\phi)$
$\tau^n_{\iota_j}(\texttt{FORALL i } \phi) = \tau^n_0(\phi) \texttt{ AND } \tau^n_1(\phi) \texttt{ AND } \ldots \texttt{ AND } \tau^n_n(\phi)$
$\tau^n(\phi) = \tau^n_{\iota_n}(\phi)$ (for any ϕ)

So the resulting formula $\tau^n(\phi)$ is equivalent to ϕ and it is made up by atoms of the form $\texttt{GRAPH } \iota_j \ \psi$ for ψ being a CQ and ι_j a constant denoting the number $j \in [n]$. As we do not have any quantification on indices the semantics becomes quite simple: We do not have to refer to a substitution σ.

We modify this fragment now in the following way: If there is a conjunction $\phi_1 \texttt{ AND } \phi_2$ with different sets variables of variables, then complete the variable on either side by adding conjunctions $x = x$ for the missing variables. This transformation does not change the semantics and also not the set of answers. Let the resulting fragment be denoted by $\mathcal{L}^{\exists,=}_{HCL}$. In order to prove the theorem, it will be enough to prove the identity for all formulas ϕ in $\mathcal{L}^{\exists,=}_{HCL}$. The idea of the proof is simple: In the $cert_{sep}$ semantics $\texttt{OR}$ and $\texttt{AND}$ act as intersection and union on the state atoms. In the $cert_h$ semantics we show that we can push $cert_h$ through to the atoms by translating $\texttt{OR}$ to union and $\texttt{AND}$ to intersection. On the atomic level $cert_{sep}$ and $cert_h$ are the same.

Proof. If $\mathcal{O} = \langle (\mathcal{A}_i)_{i\in[n]}, \mathcal{T}\rangle$ is not consistent, then we assumed that $cert_h(\phi, \mathcal{O}) = \text{NIL} = cert_{sep}(\phi, \mathcal{O})$. So in the following we may assume that $\mathcal{O}$ is consistent which means that for all $i \in [n]$ the local ontologies $\langle \mathcal{A}_i, \mathcal{T}\rangle$ are consistent.

Now we first verify that for all atoms ϕ (be it state atoms or non-state atoms) that $cert_{sep}(\phi, \mathcal{O}) = cert_h(\phi, \mathcal{O})$ holds. This is trivial if ϕ is not a state atom. If ϕ is a state atom of the form $\texttt{GRAPH } \iota_i \ \psi$ with a CQ ψ , then $cert_{sep}(\phi, \mathcal{O}) = cert_{sep}(\phi, \langle \mathcal{A}_i, \mathcal{T}\rangle) = cert(\phi, \langle \mathcal{A}_i, \mathcal{T}\rangle)$. Now, we have to show that $cert(\phi, \langle \mathcal{A}_i, \mathcal{T}\rangle) = cert_h(\phi, \mathcal{O})$. But this holds due to the definitions of satisfaction relation in the case for state atoms and due to Observation 2.

Now we show that $cert_h$ can be pushed through to the atoms by showing that it distributes over $\texttt{AND}$ and $\texttt{OR}$. The first case is that of conjunction: Let $\phi = \phi_1 \texttt{ AND } \phi_2$ where the conjuncts have the same set of free variables, say $x_1, \ldots, x_n$. Then the following chain of equalities holds: $cert_h(\phi_1 \texttt{ AND } \phi_2, \mathcal{O}) = \bigcap_{\hat{\mathcal{I}}\models\mathcal{O}} ans_{wh}(\phi_1 \texttt{ AND } \phi_2, \hat{\mathcal{I}}) = \bigcap_{\hat{\mathcal{I}}\models\mathcal{O}} ans(\phi_1, \hat{\mathcal{I}}) \cap \bigcap_{\hat{\mathcal{I}}\models\mathcal{O}} ans_{wh}(\phi_2, \hat{\mathcal{I}})$. The latter is just $cert_h(\phi_1, \mathcal{O}) \cap cert_h(\phi_2, \mathcal{O})$. The case of $\texttt{OR}$ is slightly trickier and works only due to the fact that homomorphisms preserve formulas that are in the positive existential fragment. So let be given $\phi = \phi_1 \texttt{ OR } \phi_2$ with free variables $x_1, \ldots, x_n$. Clearly we have $cert_h(\phi_1, \mathcal{O})) \cup cert_h(\phi_2, \mathcal{O})) \subseteq cert_h(\phi_1 \texttt{ OR } \phi_2, \mathcal{O})$. For the other direction assume that $\boldsymbol{c} \in cert_h(\phi_1 \texttt{ OR } \phi_2, \mathcal{O})$; and for a contradiction proof assume that $\boldsymbol{c} \notin cert_h(\phi_1, \mathcal{O}) \cup cert_h(\phi_2, \mathcal{O})$. Then there must be models $\hat{\mathcal{I}}_1, \hat{\mathcal{I}}_2 \models \mathcal{O}$ such that $\boldsymbol{c} \notin ans(\phi_1, \hat{\mathcal{I}}_1)$ and $\boldsymbol{c} \notin ans(\phi_2, \hat{\mathcal{I}}_2)$. It follows

that $\hat{\mathcal{I}}_1 \models \phi_2(\boldsymbol{c})$ and $\hat{\mathcal{I}}_2 \models \phi_1(\boldsymbol{c})$. Now we also have that $\boldsymbol{c}$ is also in the answer set for the canonical model $can(\mathcal{O})$ (see [6]), i.e., $\boldsymbol{c} \in ans(\phi_1 \texttt{ OR } \phi_2, can(\mathcal{O}))$, so either $can(\mathcal{O}) \models \phi_1(\boldsymbol{c})$ or $can(\mathcal{O}) \models \phi_2(\boldsymbol{c})$. But as $can(\mathcal{O})$ is a universal model there is a homomorphism into every model of $\mathcal{O}$, in particular for $\hat{\mathcal{I}}_1$ and $\hat{\mathcal{I}}_2$. As homomorphisms preserve positive existential clauses we must have either $\boldsymbol{c} \in ans(\phi_1, \hat{\mathcal{I}}_1)$ or $\boldsymbol{c} \in ans(\phi_2, \hat{\mathcal{I}}_2)$, contradicting our assumption. □

As a corollary to the theorem we get rewritability for STARQL queries also in the holistic semantics.

Theorem 2. *Let QL_1 be the instantiation of the* `HAVING` *clause language with ECL = UCQ and OL = DL-Lite. Then QL_1 allows for QL_1 rewriting for holistic certain query answering w.r.t. OL.*

Moreover, as `HAVING` clauses are based on FOL with $<$, `plus` (the `max` operator can be rewritten with $<$) and the state variables i can be pushed into the UCQs, we get the following additional corollary:

Corollary 1. *Let QL_1 be the instantiation of* `HAVING` *clauses with ECL = UCQ and OL = DL-Lite. Then QL_1 allows for $FOL(<, +)$ rewriting for holistic certain query answering w.r.t. OL.*

6 Comparison with LTL like Ontology Languages

The fragment of the `HAVING` clause language, for which we showed the equivalence of the holistic and the separation-based semantics, is still expressive enough to simulate some query languages that combine temporal logic operators with lightweight DL languages [2,4]. We focus on the recent query language TCQ [4]—due to the fact that in TCQ the TBox is also assumed to be non-temporal.

The query language TCQ is defined by following a weak integration of conjunctive queries (CQs) and a linear temporal logic (LTL) template.

Definition 4. *Temporal conjunctive queries (TCQs) are built from CQs as follows: each CQ is a TCQ, and if ϕ_1 and ϕ_2 are TCQs, then so are $\phi_1 \wedge \phi_2$ (conjunction), $\phi_1 \vee \phi_2$ (disjunction), $\bigcirc \phi_1$ (strong next), $\bullet \phi_1$ (weak next), $\bigcirc^- \phi_1$ (strong previous), $\bullet^- \phi_1$ (weak previous), $\phi_1 \,\mathsf{U}\, \phi_2$ (until) and $\phi_1 \,\mathsf{S}\, \phi_2$ (since).*

The satisfaction relation is defined for Boolean TCQs and on that basis the notion of answers and certain answers is defined.

Definition 5. *Let ϕ be a Boolean TCQ. For $\hat{\mathcal{I}} = (\mathcal{I}_i)_{i \in [n]}$ and $i \in [n]$ one defines $\hat{\mathcal{I}}, i \models \phi$ by induction on the structure of ϕ: $\hat{\mathcal{I}}, i \models \exists y_1, \ldots, y_m.\psi$ iff $\mathcal{I}_i \models \exists y_1, \ldots, y_m.\psi$; $\hat{\mathcal{I}}, i \models \phi_1 \wedge \phi_2$ iff $\hat{\mathcal{I}}, i \models \phi_1$ and $\hat{\mathcal{I}}, i \models \phi_2$; $\hat{\mathcal{I}}, i \models \phi_1 \vee \phi_2$ iff $\hat{\mathcal{I}}, i \models \phi_1$ or $\hat{\mathcal{I}}, i \models \phi_2$; $\hat{\mathcal{I}}, i \models \bigcirc \phi_1$ iff $i < n$ and $\hat{\mathcal{I}}, i+1 \models \phi_1$; $\hat{\mathcal{I}}, i \models \bullet \phi_1$ iff $i < n$ implies $\hat{\mathcal{I}}, i+1 \models \phi_1$; $\hat{\mathcal{I}}, i \models \bigcirc^- \phi_1$ iff $i > 0$ and $\hat{\mathcal{I}}, i-1 \models \phi_1$; $\hat{\mathcal{I}}, i \models \bullet^- \phi_1$ iff $i > 0$ implies $\hat{\mathcal{I}}, i-1 \models \phi_1$; $\hat{\mathcal{I}}, i \models \phi_1 \,\mathsf{U}\, \phi_2$ iff there is some $k, i \leqslant k \leqslant n$ s.t. $\hat{\mathcal{I}}, k \models \phi_2$ and $\hat{\mathcal{I}}, j \models \phi_1$ forall $j, i \leqslant j < k$; $\hat{\mathcal{I}}, i \models \phi_1 \,\mathsf{S}\, \phi_2$ iff there is some $k, 0 \leqslant k \leqslant i$ s.t. $\hat{\mathcal{I}}, k \models \phi_2$ and $\hat{\mathcal{I}}, j \models \phi_1$ forall $j, k < j \leqslant i$.*

The set of answers answers at i is defined as $ans(\phi, \hat{\mathcal{I}}, i) = \{\boldsymbol{a} \mid \hat{\mathcal{I}}, i \models \phi(\boldsymbol{a})\}$. *For a SO $\mathcal{O}$ the set of certain answers at i is* $cert(\phi, \mathcal{O}) = \bigcap_{\hat{\mathcal{I}} \models \mathcal{I}} ans(\phi, \hat{\mathcal{I}}, i)$. *The set of (certain) answers is defined as the (certain) answers at the last state:* $ans(\phi, \hat{\mathcal{I}}) = ans(\phi, \hat{\mathcal{I}}, n)$ *and* $cert(\phi, \mathcal{O}) := cert(\phi, \mathcal{O}, n)$.

A simple example shows that TCQ is not domain independent (though it is intended to be used in the OBDA paradigm): Take $\phi(x, y) = A(x) \vee B(y)$ and consider the interpretations $\mathcal{I} = (\{a\}, (\cdot)^{\mathcal{I}}), \mathcal{J} = (\{a, b\}, (\cdot)^{\mathcal{J}})$ with $A^{\mathcal{I}} = A^{\mathcal{J}} = \{a\}$ and $B^{\mathcal{I}} = B^{\mathcal{J}} = \varnothing$: It is $(a, b) \in ans(\phi, \mathcal{J})$ but $(a, b) \notin ans(\phi, \mathcal{I})$. Disjunction is not the only problematic constructor. Implicit disjunctions are also contained in the weak versions of the next and previous operators. Take the query $\phi(x) = \bullet A(x)$. The set of bindings to x depends on the domain *Dom*. Also, the definitions of S and U show implicit recurrence to disjunctions. Because of these reasons we consider the following safe fragment TCQ^s:

Definition 6. *TCQ^s is the fragment of TCQ where the rule for $\vee$ is replaced by: If ϕ_1, ϕ_2 are in TCQ^s and have the same free variables, then $\phi_1 \vee \phi_2$ is in TCQ^s; weak next $\bullet$ and weak previous $\bullet^-$ are disallowed; and the rules for S and U are replaced by: If ϕ_1, ϕ_2 are in TCQ^s and have the same free variables, then $\phi_1 \,\mathsf{U}\, \phi_2$ and $\phi_1 \,\mathsf{S}\, \phi_2$ are in TCQ^s.*

Now we show that every TCQ^s can be transformed into the $\mathcal{L}^{\exists}_{HCL}$ fragment of STARQL `HAVING` clauses. The translation θ is the usual one known from translating modal logics into predicate logic. It just simulates the semantics for TCQ queries within the object language of STARQL. For example, $\theta_i(\exists y_1, \ldots, y_m.\psi) =$ `GRAPH i` ψ; $\theta_i(\phi_1 \vee \phi_2) = \theta_i(\phi_1)$ `OR` $\theta_i(\phi_2)$; $\theta_i(\bigcirc^- \phi_1) =$ $i > 0$ `AND` $\theta_{i-1}(\phi_1)$.

From the similarity of the semantics for TCQ^s and the holistic semantics of $\mathcal{L}^{\exists}_{HCL}$-queries we see that the transformation θ yields for every ϕ in TCQ^s a $\mathcal{L}^{\exists}_{HCL}$-query with the same set of certain answers.

Observation 3 *For all SOs $\mathcal{O}$ and $\phi \in$ TCQ^s: $cert(\phi, \mathcal{O}) = cert_h(\theta(\phi), \mathcal{O})$*

7 Conclusion

The STARQL query language framework is part of the recent venture on lifting stream technologies to the semantic level in order to cope with the dynamic and heterogeneous data in real-world monitoring scenarios. The requirements on STARQL as a query framework to cope with OBDA and also with ABDEO paradigms lead to a specific window semantics distinguishing it from other approaches. In this paper we looked at different possibilities to specify the semantics of the `HAVING` clauses showing that both agree on a fragment capable of simulating LTL inspired query languages.

Next to the theoretical investigation of STARQL we now prepare also experiments with STARQL implementations. Besides a JAVA prototype, which is developed within the Optique platform (`http://www.optique-project.eu/`) and which can handle STARQL(DL-Lite, UCQ), we are also developing a LISP engine which can handle ABDEO instantiations of STARQL.

References

1. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. The VLDB Journal 15, 121–142 (2006)
2. Artale, A., Kontchakov, R., Wolter, F., Zakharyaschev, M.: Temporal description logic for ontology-based data access. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence. pp. 711–717. IJCAI'13, AAAI Press (2013)
3. Avron, A.: Constructibility and decidability versus domain independence and absoluteness. Theor. Comput. Sci. 394(3), 144–158 (Apr 2008)
4. Borgwardt, S., Lippmann, M., Thost, V.: Temporal query answering in the description logic DL-Lite. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) Frontiers of Combining Systems. LNCS, vol. 8152, pp. 165–180. Springer (2013)
5. Calbimonte, J.P., Jeung, H., Corcho, O., Aberer, K.: Enabling query technologies for the semantic sensor web. Int. J. Semant. Web Inf. Syst. 8(1), 43–63 (Jan 2012)
6. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodríguez-Muro, M., Rosati, R.: Ontologies and databases: The DL-Lite approach. In: Tessaris, S., Franconi, E. (eds.) 5th Int. Reasoning Web Summer School (RW 2009), LNCS, vol. 5689, pp. 255–356. Springer (2009)
7. Della Valle, E., Ceri, S., Barbieri, D., Braga, D., Campi, A.: A first step towards stream reasoning. In: Domingue, J., Fensel, D., Traverso, P. (eds.) Future Internet – FIS 2008, LNCS, vol. 5468, pp. 72–81. Springer Berlin / Heidelberg (2009)
8. Galton, A.: Reified temporal theories and how to unreify them. In: Proceedings of the 12th International Joint conference on Artificial intelligence - Volume 2. pp. 1177–1182. IJCAI'91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1991)
9. Kyzirakos, K., Karpathiotakis, M., Koubarakis, M.: Strabon: A Semantic Geospatial DBMS. In: International Semantic Web Conference. Boston, USA (Nov 2012)
10. Möller, R., Neuenstadt, C., Ozçep, O., Wandelt, S.: Advances in accessing big data with expressive ontologies. In: Timm, I.J., Thimm, M. (eds.) KI 2013: Advances in Artificial Intelligence. LNCS, vol. 8077, pp. 118–129. Springer Berlin Heidelberg (2013)
11. Özçep, Ö.L., Möller, R., Neuenstadt, C.: A stream-temporal query language for ontology based data access. In: Proceedings of the 37th German International Conference KI 2014 (2014), to be published
12. Özçep, Ö.L., Möller, R., Neuenstadt, C., Zheleznyakov, D., Kharlamov, E.: Deliverable D5.1 – a semantics for temporal and stream-based query answering in an OBDA context. Deliverable FP7-318338, EU (October 2013)
13. Phuoc, D.L., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: International Semantic Web Conference (1). pp. 370–388 (2011)
14. Valle, E.D., Schlobach, S., Krötzsch, M., Bozzon, A., Ceri, S., Horrocks, I.: Order matters! Harnessing a world of orderings for reasoning over massive data. Semantic Web 4(2), 219–231 (2013)