# A Stream-Temporal Query Language for Ontology Based Data Access[⋆]

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.** The paper contributes to the recent efforts on temporalizing and streamifiying ontology based data access (OBDA) by discussing aspects of rewritability, i.e., compilability of the TBox into ontology-level queries, and unfoldability, i.e., transformability of ontology-level queries to queries on datasource level, for the new query-language framework STARQL. The distinguishing feature of STARQL is its general stream windowing and ABox sequencing strategy which allows it to plugin well-known query languages such as unions of conjunctive queries (UCQs) in combination with TBox languages such as DL-Lite and do temporal reasoning with a sorted first-order logic on top of them. The paper discusses safety aspects under which STARQL queries that embed UCQs over DL-Lite ontologies can be rewritten and unfolded to back-end relational stream query languages such as CQL. With these results, the adoption of description logic technology in industrially relevant application areas such as industrial monitoring is crucially fostered.

**Keywords:** streams, OBDA, monitoring, unfolding, safety

## 1 Introduction

The work described in this paper is part of recent efforts on streamifying OBDA [11,6,17] and, to some extent, also temporalizing OBDA [5,4]. Streams, as potentially infinite sequences of elements, cannot be processed as a whole. Hence blocking operators such as the classical grouping operator and aggregation operators cannot be applied to it. The simple but fundamental idea of circumventing this problem is to apply on streams a (small) window the content of which is updated as new elements from the stream arrive at the query answering system.

Stream window operators play an important role also in the new query language framework STARQL (Streaming and Temporal ontology Access with a Reasoning-based Query Language, pronounced Star-Q-L, [16,15]). Its framework character relies on the facts that 1) it can embed queries of various query languages, 2) refer to ontologies in various DL languages, and 3) use a first-order

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

logic (FOL) fragment for temporal reasoning over ABox sequences constructed within the query. In this paper, we focus on the latter aspect assuming for the first two unions of conjunctive queries (UCQs) w.r.t DL-Lite ontologies.

In STARQL, the idea of processing over windows is pushed further by extending these with sequencing operators that set up at every time point a finite sequence of ABoxes on which temporal reasoning can be applied. STARQL does not assume a stream of ABoxes which hold universally but rather modifies/exploits the given ABox streams to build its own stream of finite ABox sequences. This sequencing strategy, among other things, distinguishes STARQL from the approaches in [11,6,17]. It is a natural addition to the window operators that sets up at every time point a context in which temporal reasoning can be applied.

In this paper, we consider an instantiation of STARQL where Intra-ABox reasoning within sequences is handled by answering UCQs over DL-Lite ontologies w.r.t. the certain answer semantics. Within Inter-ABox reasoning certain answers from the different ABoxes are related and constrained with an outer temporal FOL formula. This is challenging if one allows in the FOL template negation, disjunction and all quantifiers in combination with concrete domains, as these, if not constrained, would immediately lead to infinite sets of answers, in particular w.r.t. concrete domain values.

STARQL uses a new adornment technique for variables to guarantee safeness. We demonstrate the safety mechanism which will guarantee that the FOL template language is domain independent [1] and as such can be rewritten as SQL query. This opens the door for (rewriting and) unfolding STARQL queries into queries of domain independent languages such as the relational stream query language CQL [3]. Based on CQL, practical systems have been developed. Thus, this paper provides the foundation for expressive ODBA stream querying.

## 2 The STARQL framework

We describe the syntax and the semantics for a fragment of STARQL, ignoring a.o. macro definitions, aggregators etc. (see [16] for the full version; but note that here we use a different, more SPARQL like syntax). We assume familiarity with the description logic DL-Lite [7].

Our running example for illustration purposes is a measurement scenario in which there is a (possibly virtual) stream $S_{Msmt}$ of ABox assertions. Its initial part, called $S_{Msmt}^{\leq 5s}$ here, contains timestamped ABox assertions giving the value of a temperature sensor $s_0$ at 6 time points starting with $0s$.

$$\begin{aligned} S_{Msmt}^{\leq 5s} = \{ & val(s_0, 90^\circ)\langle 0s\rangle, val(s_0, 93^\circ)\langle 1s\rangle, val(s_0, 94^\circ)\langle 2s\rangle \\ & val(s_0, 92^\circ)\langle 3s\rangle, val(s_0, 93^\circ)\langle 4s\rangle, val(s_0, 95^\circ)\langle 5s\rangle\} \end{aligned}$$

Assume further, that a static ABox contains knowledge on sensors telling, e.g., which sensor is of which type. In particular, let $BurnerTipTempSens(s_0)$ be in the static ABox. Moreover, let there be a pure DL-Lite TBox with additional information such as $BurnerTipTempSens \sqsubseteq TempSens$ saying that all burner tip temperature sensors are temperature sensors.

We want to formalize the following information need: Starting with time point 0s, output every second those temperature sensors whose value grew monotonically in the last 2 seconds. A possible STARQL representation of the information is illustrated in the following listing.

```
CREATE STREAM S_out AS
CREATE PULSE AS START = 0s, FREQUENCE = 1s
CONSTRUCT  GRAPH NOW { ?s rdf:type MonInc }
FROM S_Msmt [NOW-2s, NOW]->1s , STATIC ABOX <http://Astatic>,
             TBOX <http://TBox>
WHERE { ?s rdf:type TempSens }
SEQUENCE BY StdSeq AS SEQ
HAVING FORALL i < j IN SEQ,?x,?y:
 IF (GRAPH i { ?s val ?x }  AND GRAPH j { ?s  val ?y }) THEN
     ?x <= ?y
```

Though the monotonicity condition seems simple, it should be noted that recent approaches for temporal DL-lite logics as that of [5] cannot express it.

**Syntax** The example demonstrates much of the syntactical possibilities within STARQL whose grammar is sketched in Fig. 1. The rules for the `HAVING` clause are not given there but are discussed in more detail in the following sections.

After the create expressions for the stream and the output frequency the queries' main contents are captured by the `CONSTRUCT` expressions. The head of the construct expression describes the output format of the stream, using the named-graph notation of SPARQL for fixing a basic graph pattern (BGP) and attaching a time expression, here `NOW`, for the evolving time. The general motivation for this approach is similar to the `CONSTRUCT` operator in the SPARQL query language. So the actual result in the monotonicity example (in DL notation) is a stream of ABox assertions of the form $MonInc(s_0)\langle t\rangle$.

$$S_{out}^{\leq 5s} = \{MonInc(s_0)\langle 0s\rangle, MonInc(s_0)\langle 1s\rangle, MonInc(s_0)\langle 2s\rangle, MonInc(s_0)\langle 5s\rangle\}$$

Within the `WHERE` clause one can bind variables w.r.t. the non-streaming sources (ABox, TBox) mentioned in the `FROM` clause by using (unions) of BGPs. We assume an underlying DL-Lite logic for the static ABox, the TBox and the BGP (considered as unions of conjunctive queries UCQs) which allows for concrete domain values, e.g., DL-Lite$_{\mathcal{A}}$ [7]. In this example, instantiations of the sensors $?s$ are fixed w.r.t. a static ABox and a TBox given by URIs.

The heart of the STARQL queries is the window operator in combination with the sequencing mechanism. In the example, the operator `[NOW-2s, NOW]->1s` describes a sliding window operator, which collects the timestamped ABox assertions in the last two seconds and then slides 1s forward in time. Every temporal ABox produced by the window operator is converted to a sequence of (pure) ABoxes. At every time point, one has a sequence of ABoxes on which temporal (state-based) reasoning can be applied. This is realized in STARQL by a sorted first-order logic template in which state stamped UCQs conditions are embedded. We use here again the `GRAPH` notation from SPARQL. In the example above,

$$\begin{array}{rcl}
\mathit{createExp} & \longrightarrow & \texttt{CREATE STREAM } \mathit{name} \texttt{ AS } [\mathit{pulseExp}]\ \mathit{constrExp} \mid \mathit{pulseExp} \\
\mathit{pulseExp} & \longrightarrow & \texttt{CREATE PULSE AS START = } \mathit{startTime}, \texttt{ FREQUENCE = } \mathit{freq} \\
\mathit{constrExp} & \longrightarrow & \texttt{CONSTRUCT } \mathit{constrHead}(\boldsymbol{x},\boldsymbol{y}) \\
 & & \quad \texttt{FROM } \mathit{listWinStreamExp}\ [\ \textbf{,}\ \ \mathit{listOfRessources}] \\
 & & \quad \texttt{WHERE } \mathit{whereClause}(\boldsymbol{x}) \\
 & & \quad \texttt{SEQUENCE BY } \mathit{seqMethod}\ [\texttt{HAVING } \mathit{safeHavingClause}(\boldsymbol{x},\boldsymbol{y})] \\
\mathit{constrHead}(\boldsymbol{x},\boldsymbol{y}) & \longrightarrow & \texttt{GRAPH } \mathit{timeExp}\ \mathit{BGP}(\boldsymbol{x},\boldsymbol{y})\ [\ \textbf{,}\ \ \mathit{constrHead}] \\
\mathit{listWinStreamExp} & \longrightarrow & (\mathit{name} \mid \mathit{constrExp})\mathit{windowExp}[\ \textbf{,}\ \ \mathit{listWinStreamExp}] \\
\mathit{windowExp} & \longrightarrow & \texttt{[ } \mathit{timeExp_1}, \mathit{timeExp_2} \texttt{ ]->}\mathit{sl} \\
\mathit{listOfRessources} & \longrightarrow & \mathit{typedRessourceList}[\ \textbf{,}\ \ \mathit{listOfRessources}] \\
\mathit{typedRessourceList} & \longrightarrow & \texttt{STATIC ABOX } \mathit{listofURIstoStaticABoxes} \mid \\
 & & \texttt{TEMPORAL ABOX } \mathit{listofURIstoTemporalABoxes} \mid \\
 & & \texttt{TBOX } \mathit{listofURIstoTBoxes} \\
\mathit{whereClause}(\boldsymbol{x}) & \longrightarrow & \Psi(\boldsymbol{x}) \quad (\Psi(\boldsymbol{x}) \text{ a union of BGPs with distinguished variables } \boldsymbol{x}) \\
\mathit{seqMethod} & \longrightarrow & \mathit{StdSeq} \mid \mathit{SeqMethod}(\sim)
\end{array}$$

Fig. 1: Syntax for STARQL (without `HAVING` clauses)

the `HAVING` clause expresses a monotonicity condition stating that for all values $?x$ that are values of sensor $?s$ w.r.t the $i^{th}$ ABox (subgraph) and for all values $?y$ that are values of the same sensor $?s$ w.r.t. the $j^th$ ABox (subgraph), it must be the case that $?x$ is less than or equal to $?y$.

**Semantics** STARQL queries have streams of ABox assertions (RDF triples) as input and output. So, the semantics for STARQL has to explicate how the output stream of ABox assertions is computed from the input streams. Using compositionality, the semantics definition for STARQL can be accomplished by defining the semantic denotations for the substructures of the query and then by composing them to the denotation of the whole query. We sketch the semantics of the window operator, of the sequencing, and of the `HAVING` clause.

A stream of ABox assertions is an infinite set of timestamped ABox assertions of the form $ax\langle t\rangle$. The timestamps stem from a flow of time $(T, \leq)$ where $T$ may even be a dense set and where $\leq$ is a linear order. Let $S$ be a stream name with its denotation $[\![S]\!]$ being such a stream of timestamped ABox assertions. We declare the denotation of the windowed stream $ws = S\ winExp = S$ `[`$\mathit{timeExp_1}, \mathit{timeExp_2}$`]->`$sl$ as a stream of temporal ABoxes, where a temporal ABox is a set of timestamped assertions.

Let $\lambda t.g_1(t) = [\![\mathit{timeExp}_1]\!]$ and $\lambda t.g_2(t) = [\![\mathit{timeExp}_2]\!]$ be the functions corresponding to the time expressions. The pulse declaration defines a subset $T' \subseteq T$ of the time domain. $T'$ is the set of timestamps of the stream of temporal ABoxes

$[\![ws]\!]$. 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.

Now, one defines for every $t_i$ the temporal ABox $\tilde{\mathcal{A}}_{t_i}$ such that $(\tilde{\mathcal{A}}_{t_i}, t_i) \in [\![wS]\!]$. If $t_i < sl - 1$, then $\tilde{\mathcal{A}}_{t_i} = \emptyset$. 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

$$\tilde{\mathcal{A}}_{t_i} = \{(ass, t) \mid (ass, t) \in [\![S]\!] \text{ and } t_{end} \leq t \leq t_{start}\}$$

In our example, $timeExp_1 = NOW - 2s$ (so $[\![timeExp_1]\!] = \lambda t.t - 2$), $timeExp_2 = NOW$ and $sl = 1s$; the example's results for second 4s and 5s are the following.

| Time | Temporal ABox |
|---|---|
| $4s$ | $\{val(s_0, 94°)\langle 2s\rangle, val(s_0, 92°)\langle 3s\rangle, val(s_0, 93°)\langle 4s\rangle\}$ |
| $5s$ | $\{val(s_0, 92°)\langle 3s\rangle, val(s_0, 93°)\langle 4s\rangle, val(s_0, 95°)\langle 5s\rangle\}$ |

If the STARQL query refers to more than one stream, then these are joined by time-wise union of the temporal ABoxes of the windowed streams, which is possible as the pulse declaration synchronizes all streams of temporal ABoxes.

The stream of temporal ABoxes is the input for the sequencing operator which produces for every time point of the pulse a sequence of (pure) ABoxes. 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, i.e., the equivalence classes under $\sim$ should be intervals on the time domain.

Now, we define the sequence of ABoxes generated by $seqMethod(\sim)$ on the stream of temporal ABoxes as follows: Let $(\tilde{\mathcal{A}}_t, t)$ be the temporal ABox 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 $i \in \{0, \ldots, k\}$ the pure ABox $\mathcal{A}_i$ is

$$\mathcal{A}_i = \{ax\langle t'\rangle \mid ax\langle t'\rangle \in \tilde{\mathcal{A}}_t \text{ and } t' \text{ in } i^{th} \text{ equivalence class}\}$$

In the example above, the equivalence is the identity (keyword `StdSeq` for standard sequencing), so that the resulting sequence of ABoxes at time point 5s is trivial as there are no more than two ABox assertions with the same timestamp: $\{val(s_0, 92°)\}\langle 0\rangle, \{val(s_0, 93°)\}\langle 1\rangle, \{val(s_0, 95°)\}\langle 2\rangle$.

STARQL's semantics for the `HAVING` clauses relies on the certain answer semantics (see [7]) for the embedded UCQs. The idea is to view the tuples in the certain answer sets as members of a sorted FOL structure $\mathcal{I}_t$. Assume that the sequence of ABoxes at some given time point $t$ is $seq = (\mathcal{A}_0, \ldots, \mathcal{A}_k)$. Then the domain of $\mathcal{I}_t$ consists of the index set $\{0, \ldots, k\}$ as well as the set of all individual constants and all value constants of the signature. Now, if the `HAVING` clause contains, for example, the state tagged condition query $val(s, x)\langle i\rangle$ (with embedded UCQ $val(s, x)$), then we introduce for it a ternary relation symbol $R$ and replace $val(s, x)\langle i\rangle$ by $R(s, x, i)$ in the `HAVING` clause. This symbol is denoted in $\mathcal{I}_t$ by the certain answers of the embedded query extended with the index $i$:

$R^{\mathcal{I}_t} = \{(a, b, i) \mid (a, b) \in cert(val(s, x), \mathcal{A}_i \cup \mathcal{A}_{static} \cup \mathcal{T})\}$. 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 $\mathcal{I}_t$.

## 3 A Safe Fragment for `HAVING` clauses

As demonstrated with the monotonicity example, STARQL allows a sorted FOL to reason on the ABox sequences. The semantics for the `HAVING` clauses rests on the structure $\mathcal{I}_t$ whose domain $\Delta^{\mathcal{I}_t}$ does not consist only of the individual and value constants in the interpretations of the relations, the so called *active domain* according to database terminology [1], but the whole set *Dom* of individual constants, value constants and the indices produced in the sequencing. With a safety mechanism on the `HAVING` clauses it can be guaranteed that the evaluation of the `HAVING` clause on the ABox sequence depends only on the active domain, i.e., `HAVING` clauses are domain independent (d.i.). Formally, a query $q$ is d.i. iff for all interpretations $\mathcal{I}_1, \mathcal{I}_2$ having domains $\Delta^{\mathcal{I}_1}, \Delta^{\mathcal{I}_2} \subseteq Dom$ and identical denotation functions $(\cdot)^{\mathcal{I}_1} = (\cdot)^{\mathcal{I}_2}$, the answers for $q$ in $\mathcal{I}_1$ is the same as the answers for q in $\mathcal{I}_2$. Without a safety mechanism, a `HAVING` clause of the form $y > 3$, with free concrete domain variable $y$, would be allowed: the set of bindings for $y$ would be infinite, namely, the set of all real number bigger than 3. In particular, $y > 3$ is not d.i.

Figure 2 contains the grammar for the `HAVING` clauses with its safety mechanism realized by variable guards/adornments. The safe `HAVING` clauses (denoted by the start symbol *safeHavingClause*) contain only those variables for individuals that have guard status $+$. We illustrate the meaning of the rules with Rule (1) for the `OR` case and then go in more detail w.r.t. adornments.

$$hCl(\boldsymbol{z}^{\boldsymbol{g}^1 \vee \boldsymbol{g}^2}) \longrightarrow hCl(\boldsymbol{z}^{\boldsymbol{g}^1}) \texttt{ OR } hCl(\boldsymbol{z}^{\boldsymbol{g}^2}) \tag{1}$$

A having clause $hCl$ may be constructed as (produces) a disjunction of two having clauses under some conditions on the variables occurring in them. If during production a clause $hCl(\boldsymbol{z}^{\boldsymbol{g}})$ with variables $\boldsymbol{z}$ and some adornment $\boldsymbol{g}$ for them is reached, then Rule (1) justifies the production of $hCl(\boldsymbol{z}^{\boldsymbol{g}^1})$ `OR` $hCl(\boldsymbol{z}^{\boldsymbol{g}^2})$ if the adornment $\boldsymbol{g}$ can be represented as $\boldsymbol{g} = \boldsymbol{g}^1 \vee \boldsymbol{g}^2$, i.e., if $\boldsymbol{g}$ is the result of applying a function $\vee$ on the adornment lists $\boldsymbol{g}^1, \boldsymbol{g}^2$.

The adornments $\boldsymbol{g} = g_1, \ldots, g_n$ are lists of guard status $g_i$ (g-status for short), where $g_i \in \{+, -, --, \emptyset\}$. We use $\boldsymbol{z}^{\boldsymbol{g}}$ as an abbreviation for $z_1^{g_1}, \ldots, z_n^{g_n}$ where $\boldsymbol{z} = z_1, \ldots, z_n$ and $\boldsymbol{g} = g_1, \ldots, g_n$. We assume the ordering $\emptyset \preceq -- \preceq - \preceq +$ on the guards. This ordering is relevant for the calculation of $g_{max}$ in the rule of Fig. 2 where the clause is constructed from an arbitrary clause $hCl$ and an identity atom. The special case of $g_i = \emptyset$ is a convenience notation meaning for $x^{\emptyset}$ that $x$ does not occur at all in the formula.

The meanings of the functions $\neg, \vee, \wedge, \rightarrow$ over vectors of g-status are fixed by the tables in Figure 3. Combinations with the g-status $\emptyset$ is handled in

$$
\begin{array}{rl}
\mathit{safeHavingClause}(\boldsymbol{z}) & \longrightarrow \mathit{hCl}(\boldsymbol{z}^{+}) \quad (\text{for } \boldsymbol{z} \in Var_{val} \cup Var_{ind}) \\
\\
\mathit{term}(i^{+}) & \longrightarrow i \\
\mathit{term}() & \longrightarrow \texttt{max} \mid \texttt{0} \mid \texttt{1} \\
\mathit{stateAtom}(\boldsymbol{y}^{+}, i^{+}) & \longrightarrow \texttt{GRAPH i } \Psi(\boldsymbol{x}, \boldsymbol{y}) \\
 & \phantom{\longrightarrow} (\text{for a UCQ } \Psi(\boldsymbol{x}, \boldsymbol{y}) \text{ and} \\
 & \phantom{\longrightarrow} \boldsymbol{x} \subseteq X,\ \boldsymbol{y} \subseteq Var_{ind} \cup Var_{val} \setminus X) \\
\mathit{stateAtom}(x^{--}, y^{--}) & \longrightarrow x = y \quad (\text{for } y, x \notin X \cup Var_{val}) \\
\mathit{stateAtom}(x^{+}) & \longrightarrow x = a \mid a = x \\
 & \phantom{\longrightarrow} (\text{for } a \in (X \cap Var_{ind}) \cup Const_{const},\ x \in Var_{ind} \setminus X) \\
\mathit{vAtom}(z_1^{+}) & \longrightarrow z_1 = v \mid v = z_1 \\
 & \phantom{\longrightarrow} (\text{for } z_1 \in Var_{val} \setminus X \text{ and } v \in Const_{val}) \\
\mathit{vAtom}(z_1^{+}) & \longrightarrow z_1 = z_2 \mid z_2 = z_1 \\
 & \phantom{\longrightarrow} (\text{for } z_1 \in Var_{val} \setminus X, \text{ and } z_2 \in X \cap Var_{val}) \\
\mathit{vAtom}(z_1^{--}, z_2^{--}) & \longrightarrow z_1 \ \mathit{op} \ z_2 \\
 & \phantom{\longrightarrow} (\text{for } \mathit{op} \in \{\texttt{<,<=, >, >=, =}\};\ z_1, z_2 \in Var_{val} \setminus X) \\
\mathit{vAtom}(z_1^{--}) & \longrightarrow z_1 \ \mathit{op} \ z_2 \quad (\text{for } \mathit{op} \in \{\texttt{<,<=, >, >=}\},\ z_1 \in Var_{val} \setminus X, \\
 & \phantom{\longrightarrow z_1 \ \mathit{op} \ z_2 \quad (\text{for }} z_2 \in Val_{const} \cup (X \cap Val_{var}) \\
\mathit{stateArithAtom}(i_1^{g_1}, i_2^{g_2}) & \longrightarrow \mathit{term}_1(i_1^{g_1}) \ \mathit{op} \ \mathit{term}_2(i_2^{g_2}) \\
 & \phantom{\longrightarrow} (\text{for } \mathit{op} \in \{\texttt{<,<=, =, >, >=}\}) \\
\mathit{stateArithAtom}(i_1^{g_1}, i_2^{g_2}, i_3^{g_3}) & \longrightarrow \texttt{plus(}\mathit{term}_1(i_1^{g_1}), \mathit{term}_2(i_2^{g_2}), \mathit{term}_3(i_3^{g_3})\texttt{)} \\
\\
\mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}}) & \longrightarrow \mathit{stateAtom}(\boldsymbol{z}^{\boldsymbol{g}}) \mid \mathit{vAtom}(\boldsymbol{z}^{\boldsymbol{g}}) \mid \mathit{stateArithAtom}(\boldsymbol{z}^{\boldsymbol{g}}) \\
\mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^1 \vee \boldsymbol{g}^2}) & \longrightarrow \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^1}) \ \texttt{OR} \ \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^2}) \\
\mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^1 \wedge \boldsymbol{g}^2}) & \longrightarrow \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^1}) \ \texttt{AND} \ \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^2}) \quad (\text{both conjuncts are} \\
 & \phantom{\longrightarrow} \text{not of form } x = y \text{ for } x, y \in Var_{ind} \cup Var_{val}) \\
\mathit{hCl}(z_1^{g_{max}}, z_2^{g_{max}}, \boldsymbol{z_3}^{g_3}) & \longrightarrow \mathit{hCl}(z_1^{g_1}, z_2^{g_2}, \boldsymbol{z_3}^{g_3}) \ \texttt{AND} \ z_1^{h_1} = z_2^{h_2} \\
 & \phantom{\longrightarrow} (\text{for } g_{max} = max\{g_1, g_2, h_1, h_2\}) \\
\mathit{hCl}(\boldsymbol{z}^{\neg \boldsymbol{g}}) & \longrightarrow \texttt{NOT} \ \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}}) \\
\mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^1 \rightarrow \boldsymbol{g}^2}) & \longrightarrow \texttt{IF} \ \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^1}) \ \texttt{THEN} \ \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^2}) \\
\mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^1 \rightarrow \boldsymbol{g}^2}) & \longrightarrow \texttt{FORALL} \ \boldsymbol{y} \ \texttt{IF} \ \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^1}, \boldsymbol{y}^{+}) \ \texttt{THEN} \ \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^2}, \boldsymbol{y}^{g}) \\
\mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^1 \wedge \boldsymbol{g}^2}) & \longrightarrow \texttt{EXISTS} \ \boldsymbol{y} \ \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^1}, \boldsymbol{y}^{+}) \ \texttt{AND} \ \mathit{hCl}(\boldsymbol{z}^{\boldsymbol{g}^2}, \boldsymbol{y}^{g})
\end{array}
$$

Fig. 2: Grammar for `HAVING` clauses
(the set of variables $X$ is the set of variables that are bounded by the `WHERE` clause in the STARQL query)

an extra table 3b. So for example, assume that one has produced a `HAVING` clause $F(x_1^{--}, x_2^{+}, x_3^{-})$, where $x_1$ has g-status $--$, $x_2$ has g-status $+$, and $x_3$ has g-status $-$. Then rule (1) and the tables allow, e.g., the production of $F_1(x_1^{--}, x_2^{+}, x_3^{-})$ `OR` $F_2(x_1^{+}, x_2^{+}, x_3^{\emptyset})$. Let us verify this for the variable $x_1$: Its g-status $--$ in $F_1$ and its g-status $+$ in $F_2$ combines to the g-status $-- = -- \vee +$ in F—according to the entry for the pair $(--, +)$ in the table of $\vee$.

| $g_1$ | $g_2$ | $\neg g_1$ | $g_1 \wedge g_2$ | $g_1 \vee g_2$ | $g_1 \rightarrow g_2$ |
|---|---|---|---|---|---|
| $--$ | $--$ | $--$ | $--$ | $--$ | $--$ |
| $--$ | $-$ | $--$ | $--$ | $-$ | $-$ |
| $--$ | $+$ | $--$ | $+$ | $--$ | $--$ |
| $-$ | $--$ | $+$ | $--$ | $-$ | $--$ |
| $-$ | $-$ | $+$ | $-$ | $-$ | $-$ |
| $-$ | $+$ | $+$ | $+$ | $-$ | $+$ |
| $+$ | $--$ | $-$ | $+$ | $--$ | $-$ |
| $+$ | $-$ | $-$ | $+$ | $-$ | $-$ |
| $+$ | $+$ | $-$ | $+$ | $+$ | $-$ |

(a) Variables existent in both subformulas

| $g_1$ | $g_2$ | $g_1 \wedge g_2$ | $g_1 \vee g_2$ | $g_1 \rightarrow g_2$ |
|---|---|---|---|---|
| $--$ | $\emptyset$ | $--$ | $--$ | $--$ |
| $-$ | $\emptyset$ | $-$ | $-$ | $--$ |
| $+$ | $\emptyset$ | $+$ | $--$ | $-$ |
| $\emptyset$ | $--$ | $--$ | $--$ | $--$ |
| $\emptyset$ | $-$ | $-$ | $-$ | $-$ |
| $\emptyset$ | $+$ | $+$ | $--$ | $--$ |

(b) Variable missing in one subformula

Fig. 3: Combination of Guards

Now, we will show how to transform `HAVING` clauses to SQL. In particular this shows that the `HAVING` clause language is d.i. as SQL is d.i.For a formula $F$ let $SRNF(F)$ be the formula in *safe range normal form (SRNF)* [1, S.85] resulting from applying the following normalization steps: Rename variables such that no variable symbol occurrence is bound by different quantifiers and such that no variable occurs bound and free; rewrite `IF` $F$ `THEN` $G$ to `NOT` $F$ `OR` $G$; eliminate double negations; rewrite `FORALL`$z$ with `NOT EXISTS z NOT`; push `NOT` through using de Morgan rules. These steps are applied in some order until they cannot be applied anymore. A formula $F$ is said to be in SRNF iff $F = SRNF(F)$.

Domain independence for formulas in SRNF is handled in the literature [1] also by a guard concept. This is realized by a function $rr$ as follows.

1. $rr(\mathtt{r}(t_1, \ldots, t_n))$ = variables in $t_1, \ldots, t_n$.
2. $rr(x\ op\ y) = \emptyset$ for $x, y \in Var_{val}, op \in \{<, >, \leq, \geq\}$
3. $rr(x\ op\ v) = rr(x\ op\ v) = \emptyset$ for $x \in Var_{val}, v \in Const_{val}, op \in \{<, >, \leq, \geq\}$
4. $rr(x = a) = rr(a = x) = \{x\}$ (for $x \in Var,\ a \in Const$)
5. $rr(F$ `AND` $G) = rr(F) \cup rr(G)$
6. $rr(F$ `AND` $(x = y)) = \begin{cases} rr(F) \cup \{x, y\} & \text{if } rr(F) \cap \{x, y\} \neq \emptyset \\ rr(F) & \text{else} \end{cases}$
7. $rr(F$ `OR` $G) = rr(F) \cap rr(G)$
8. $rr($`NOT` $F) = \emptyset$
9. $rr($`EXISTS` $\boldsymbol{x}F) = \begin{cases} rr(F) \setminus \boldsymbol{x} \text{ if } \boldsymbol{x} \subseteq rr(F) \\ \text{return } \perp \text{ else} \end{cases}$

The definition of $rr$ in [1] are simpler than our adornment technique used in the grammar because of two main reasons: the authors in [1] assume that the formula is already in SRNF form, whereas we do not. Moreover, we define the `HAVING` clause grammar in the context of the grammar for STARQL queries. So, we have to take care of variables $X$ that are already bounded by the `WHERE` clause. This leads to many sub-cases in our grammar.

A formula $F$ in SRNF is called *range restricted* iff $free(F) = rr(F)$ and no subformula returns $\bot$. A well-known theorem states that range restricted formulas in SRNF are exactly as expressive as relational algebra—which is known to be d.i. Hence it is well-known that safe range formulas are d.i. (in particular all sets of answers are finite).

Relating our set of g-status with the set of g-status used in [1] leads to the desired theorem.

**Theorem 1.** *All safe `HAVING` clauses (considered as queries on the DB $\mathcal{I}_t$ of certain answers within the actual ABox sequence at t) are d.i.*

Let $safehCl(\boldsymbol{u}^+)$ be a safe `HAVING` clause. Let $safehClNF(\boldsymbol{u})$ be the formula resulting from applying the SRNF normalization rules. The status of all the guards are not changed by the rules. Now, we see that for all subformulas $G(\boldsymbol{x}^+, \boldsymbol{y}^-, \boldsymbol{z}^{--})$ in $safehClNF(\boldsymbol{u})$ we have

**(*)** $rr(G) = \boldsymbol{x}$ (= all variables in $G$ with g-status +)

The proof of (∗) is by structural induction on construction of the formula $safehcLNF(\boldsymbol{u})$. Let $G(\boldsymbol{x}^+, \boldsymbol{y}^-, \boldsymbol{z}^{--})$ be an atomic clause. Then $rr(G) = \boldsymbol{x}$ follows from looking at the adornments of the atomic clauses $G$ in Fig. 2 and checking that only those with g-status + are in $rr(G)$. Hereby, variables $x \in X$, where $X$ is defined in the grammar, are treated as constants in the definition of $rr(\cdot)$. The case of conjunction is clear too as any + g-status combines with any other g-status to +. Now take negation $G = \texttt{NOT}\ F(\boldsymbol{x}^+, \boldsymbol{y}^-, \boldsymbol{z}^{--})$. The definition of $rr$ for the negation case says $rr(G) = \emptyset$. Actually we know that $F$ is an atomic formula. Looking at all variables for these formulas in the grammar we see that no one of these as g-status −, hence actually $\boldsymbol{y} = \emptyset$ and we have $G(\boldsymbol{x}^-, \boldsymbol{z}^{--})$, so there is no variable in $G$ with g-status +, hence indeed we get that $rr(G) = \emptyset =$ the variables in $G$ with g-status +. The case for disjunction is clear as a positive g-status results for a variable in a disjunction only if both variables exists in the disjuncts and are labelled +. Now the last case is that of the existential quantifier $G = \texttt{EXISTS}\ xF(\boldsymbol{x}^+, \boldsymbol{y}^-, \boldsymbol{z}^{--})$. According to induction assumption $rr(F) = \boldsymbol{x}$. $G$ may result from a transformation of an exists subformula `EXISTS` $xhCl(x^+, \dots)$ `AND` $F'$ in $hcl(\boldsymbol{u}^+)$. So the variable $x$ is by definition in the set $\boldsymbol{x}$ of variables in $F$ with g-status +, hence $rr(G) = rr(F) \setminus \{x\}$. But $G$ does not occur as free variable in $G$, hence the set of variables in $G$ with g-status + is actually $\boldsymbol{x}$ without $x$, which proves the induction claim. Now $G$ may also result from applying somewhere the rule `FORALL` $\equiv$ `NOT EXISTS NOT`. But again, there is a formula with variables that have g-status + and are bounded by the all quantifier so that one gets again a formula of the form `EXISTS` $xhCl(x^+, \dots)$ `AND` $F'$.

## 4 Unfolding STARQL into CQL

Having shown domain independence for the `HAVING` clause language is the main step towards using STARQL for OBDA in the classical sense according to which queries on the ontology level are rewritten and unfolded into queries over the data source. The general procedure is illustrated below.

```
Input: STARQL Query SQ, Mappings M      Output: CQL query OQ
SQ1 = Rewrite(SQ, TBox(SQ))
SQ2 = SRNF(SQ1)
OQ = Unfold(SQ2, M)
```

Rewriting is done locally w.r.t. every embedded UCQ, using the TBox of the STARQL query `SQ`. The result `SQ1` is transformed to a query `SQ2` in (range-restricted) SRNF, which can be unfolded to an SQL like streaming language query `OQ`. In the following we illustrate the unfolding process into CQL queries.

CQL [3] is one of the early relational stream query languages having served as a blue print for many stream query languages even on the ontological level (e.g., [11],[6],[17]). CQL window operators get as input a stream and produce a temporal relation, which is a function over the time domain $T$ giving for every $t$ an ordinary (instantaneous) relation $R_t$. The operator *RStream* gets a temporal relation $R$ as input and produces a stream of tuples $d\langle t\rangle$ such that $d \in R_t$.

Following the classical OBDA approach we assume that the streams to which STARQL refers are produced by mappings. In our example, let be given a CQL stream of measurements $Msmt$ with schema `Msmt(MID, MtimeStamp, SID, Mval)`. A mapping takes a CQL query over this stream and produces a stream of assertions of the form $val(x, y)\langle t\rangle$.

$val(x, y)\langle z\rangle \longleftarrow$ `SELECT Rstream(f(SID) as x, Mval as y,`
`MtimeStamp as z) FROM Msmt[NOW]`

We assume that the STARQL queries use the standard sequencing only, so that from every state $i$ in the sequence associated with $t_{NOW}$ one can reconstruct the timestamps of the tuples occurring in the ABox $\mathcal{A}_i$.

The following listing shows the unfolded CQL pendant of the STARQL query. The outer `WHERE` clause is the pendant (in SRNF form) of the monotonicity formula expressed in the `HAVING` clause.

```
CREATE VIEW windowRelation as
SELECT *  FROM Msmt[RANGE 2s Slide 1s];
SELECT Rstream(sensor, timestamp)
FROM windowRel, sensorRel
WHERE sensorRel.type = ''BurnerTipTempSens'' AND
NOT EXISTS (
 SELECT * FROM
 (SELECT timestamp as i, value as x FROM windowRelation),
 (SELECT timestamp as j, value as y FROM windowRelation)
 WHERE   i < j AND x > y );
```

## 5 Related Work

Much of the relevant work on stream processing has been done in the context of data stream management systems (DSMSs), mainly with SQL-like stream query languages such as CQL [3] or the ones used in TelegraphCQ [8], Aurora/Borealis [12], or PIPES [14]. Nevertheless, the stream community is far from having a query standard for DSMS (see [13] for some ideas).

First steps towards streamified OBDA are C-SPARQL [11], SPARQLstream [6], and CQELS [17]. These approaches extend SPARQL with a window operator whose content is a multi-set of variable bindings for the open variables in the query. This solution is not without problems. It presupposes mixed interim states in which the constraints/consequences of the ontologies are not accounted for out. In particular, the window operator's forgetfulness w.r.t. time stamps lead to inconsistencies that are not in the input streams.

The semantical foundation of evaluating `HAVING` clauses is similar to that of [5], one of the recent approaches to temporalizing OBDA. (Another one is [4]). The difference is that [5] uses an LTL based language with embedded CQs not a sorted FOL language. For engineering applications with information needs as in the monotonicity example the LTL framework is not sufficient, as it does not provide existential quantifiers on top of the embedded CQs.

Safety conditions are considered in the classical DB literature [1] but also specifically for temporal DBs [9]. We do not claim novelty w.r.t. safety aspects but only w.r.t. the new adornment technique which directly operates on the FOL formulas without transforming them to some normal form.

Though not directly related to OBDA, other relevant work stems from the field of *complexed event processing*. For example, EP-SPARQL/ETALIS [2] uses also a sequencing constructor; and T-REX with the event specification language TESLA [10] uses an FOL language for identifying patterns.

## 6 Conclusion

The paper has presented a query framework lying in the intersection of classical OBDA and stream processing. The query language (necessarily) extends the sliding window concepts, which are known from many languages for relational stream data management systems as well as recent systems for RDFS, with ABox sequencing constructors. The advantage of using a sequence based methodology over other approaches are, first, that the sequence sets up a (nearly) standard context in which standard OBDA reasoning services can be applied, and second, that the query language can be equipped with a neat semantics based on the certain answer semantics for pure DL-Lite ABoxes (see [16]).

STARQL's combination of sufficient expressiveness on the conceptual level with high expressiveness w.r.t. arithmetical, and statistical computations as well as event specifications can be implemented in a safe manner in order to reach domain independence. This lays the ground for a complete and correct transformation to streaming query languages on the backend data sources.

## References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Stream reasoning and complex event processing in ETALIS. Semantic Web 3(4), 397–407 (2012)
3. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. The VLDB Journal 15, 121–142 (2006)
4. Artale, A., Kontchakov, R., Wolter, F., Zakharyaschev, M.: Temporal description logic for ontology-based data access. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. pp. 711–717. IJCAI'13 (2013)
5. 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)
6. 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)
7. 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.) Semantic Technologies for Informations Systems (RW 2009), LNCS, vol. 5689, pp. 255–356. Springer (2009)
8. Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.A.: TelegraphCQ: Continuous dataflow processing for an uncertain world. In: CIDR (2003)
9. Chomicki, J., Toman, D.: Temporal databases. In: Handbook of Temporal Reasoning in Artificial Intelligence, vol. 1, pp. 429–467. Elsevier (2005)
10. Cugola, G., Margara, A.: TESLA: A formally defined event specification language. In: Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems. pp. 50–61. DEBS '10, ACM, New York, NY, USA (2010)
11. 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, Lecture Notes in Computer Science, vol. 5468, pp. 72–81. Springer Berlin / Heidelberg (2009)
12. Hwang, J.H., Xing, Y., Çetintemel, U., Zdonik, S.B.: A cooperative, self-configuring high-availability solution for stream processing. In: ICDE. pp. 176–185 (2007)
13. Jain, N., Mishra, S., Srinivasan, A., Gehrke, J., Widom, J., Balakrishnan, H., Çetintemel, U., Cherniack, M., Tibbetts, R., Zdonik, S.: Towards a streaming SQL standard. Proc. VLDB Endow. 1(2), 1379–1390 (Aug 2008)
14. Krämer, J., Seeger, B.: Semantics and implementation of continuous sliding window queries over data streams. ACM Trans. Database Syst. 34(1), 1–49 (Apr 2009)
15. Özçep, O.L., Möller, R., Neuenstadt, C.: Obda stream access combined with safe first-order temporal reasoning. Techn. report, Hamburg Univ. of Technology (2014)
16. 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)
17. 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)