OBDA Stream Access Combined with Safe First-Order Temporal Reasoning

Özgür L. Özçep and Ralf Möller and Christian Neuenstadt

28.02.2014

Abstract

Stream processing is a general information processing paradigm with different applications in AI. Most stream languages rely on the concept of a sliding window with a bag semantics, which is in order for relational streams but may lead to inconsistencies when applied on streams of assertions evaluated against a concep- tual model. Our approach uses a different semantics based on ABox sequencing. The query language provides an expressive first order temporal logic for inter-ABox reasoning. Safety conditions tame the expressiveness so that a meaning preserving transformation of the query to backend queries on the sources as foreseen in the OBDA paradigm is guaranteed.

OBDA Stream Access Combined with Safe First-Order Temporal Reasoning (Extended Version)

Özgür L. Özçep[1] and **Ralf Möller**[1] and **Christian Neuenstadt**[1]

Abstract. Stream processing is a general information processing paradigm with different applications in AI. Most stream languages rely on the concept of a sliding window with a bag semantics, which is in order for relational streams but may lead to inconsistencies when applied on streams of assertions evaluated against a conceptual model. Our approach uses a different semantics based on ABox sequencing. The query language provides an expressive first order temporal logic for inter-ABox reasoning. Safety conditions tame the expressiveness so that a meaning preserving transformation of the query to backend queries on the sources as foreseen in the OBDA paradigm is guaranteed.

1 Introduction

Data processing over streams—an ever extended set of time-tagged data—is a general information processing paradigm with realizations in different AI applications such as understanding narratives on top of NLP, interpretation of video streams [9], complex event processing [2], etc. Early relational stream data management systems (RSDMS) such as [3] rely on the concept of a sliding window, which is used to, first, collect data from the stream as far as allowed by the window size, second do calculations of the contents, and then move forward in time to incorporate new incoming data. The window concept also has become important in recent RDF stream processing systems such as C-SPARQL [18], which provides online incremental reasoning w.r.t. RDFS+, or SPARQLstream [5], which provides first ideas on accessing data streams over ontologies and mappings according to the OBDA paradigm [6]. In OBDA queries on the ontological level can be transformed/reduced to queries on the backend data sources—thereby preserving meaning.

All recent systems for processing streams using conceptual models have in common that they use a " bag of assertions" semantics for the window, whereby all incoming assertions in a stream (independently of there time tag) are put into a bag. Though this is a direct adaptation of the window semantics for relational streams as explicated by [3], it has to be noted that this loss of temporal information bears problems due to potential logical inconsistencies that can occur w.r.t. the conceptual model.

Also because of this reason, the query language STARQL (pronounced Star-Q-L), which we develop in this paper, implements a different semantics based on the general concept of ABox sequencing. Assertions in a window are depending on the time stamps and the chosen sequencing strategy grouped into ABoxes, the result being a sequence of ABoxes (at every time point). The semantics of processing the ABox sequencing has two aspects. The first, intra-ABox OBDA, is a nearly classical OBDA for each ABox; in fact, STARQL embeds classical SPARQL queries (`http://www.w3.org/TR/rdf-sparql-query/`) within its grammar definition. However, different from classical OBDA, each ABox is dynamically constructed from a stream and may have to be combined with possibly huge (static) ABoxes modeling non-temporal knowledge. The second aspect, inter-ABox reasoning, combines the outcomes of the first component and applies some form of temporal reasoning. Both aspects are mainly covered by the highly expressive sublanguage within STARQL.

In this paper, we present a result related to the second aspect of the STARQL semantics. The overall aim is to guarantee that every STARQL query can be transformed into, e.g., a CQL query [3] over the backend sources. Intra-ABox OBDA is not problematic w.r.t. the rewriting of queries, i.e., w.r.t. compiling the intensional knowledge of the TBox into the query. But for the unfolding of the queries (mapping the rewritten queries to queries in the language of the backend sources) one cannot rely on the unfolding mechanisms for classical OBDA, i.e., here intra-ABox OBDA and inter-ABox are not separable. In order to guarantee unfoldability, the highly expressive first order logic used in query clauses requires the use of a safety mechanism, which in essence will guarantee the fulfilment of the so-called well known domain independence property. Considering this property, the STARQL framework can be considered to be a contribution for integrating stream processing into many AI applications.

The article is structured as follows. We first discuss related work to motivate the query language introduced afterwards. With some examples from sensor measurement scenarios we motivate the main features of this language. We define a suitable semantics of a sublanguage of STARQL for inter-ABox clauses, and demonstrate a safety mechanism to guarantee domain independence. The appendix contains the syntax and the semantics of STARQL as well as the proof of the main theorem. Related Work We focus our discussion on *stream-based data processing* literature, and assume that sensor data is transferred to a computer center. From a database perspective, with data passed from sensors to a central computer system, many stream related investigations by different research groups have been pursued in the period from 2003 to 2008. Though not directly part of AI research, many of the fundamental concepts such as that of a window are used in more AI/knowledge representation oriented stream systems such as C-SPARQL, SPARQLstream etc. There exist different data stream management systems (DSMSs), mainly with SQL-like stream query languages such as, for instance, CQL [3]. The languages are semantically well-founded and systems have been tested with well-defined benchmarks. Besides theoretical work and academic prototypes, such as STREAM [3], TelegraphCQ[7], Aurora/Borealis [10], or PIPES [13], one can also see various commercial systems, either standalone systems such as StreamBase, Truviso, or

[1] Institute for Software Systems, Hamburg University of Technology, Germany, email: {oezguer.oezcep, moeller, christian.neuenstadt}@tu-harburg.de

stream extensions to commercial relational DBMSs (MySQL, PostgreSQL, DB2 etc.). Nevertheless, as of now, the stream community is far from having a query standard for DSMS (see [11] for some ideas).

In the description logic (DL) community, stream-based data processing w.r.t. ontologies was first mentioned in [8]. For a recent stream processing system using DL see [9]. Stream-based processing has also been investigated w.r.t. the RDF data model. Prominent systems are C-SPARQL [18] (developed within LarKC, `www.larkc.eu`), SPARQLstream [5], or CQELS [16]. For querying time-tagged triples all systems use extensions to SPARQL. In the spirit of CQL these systems use some "window" concept to support reactive data processing with continuous queries. C-SPARQL introduces the notion of "stream reasoning", referring to answering window-based continuous queries w.r.t. RDFS+ ontologies. All of theses approaches rely on the bag-semantics for windows, which as mentioned in the introduction, may cause problems w.r.t. a conceptual model. Consider, e.g., a model (a TBox in DL speak) stating that a sensor shows at one time point at most one value. Hence assertions from different time points with different values for the same sensor may contradict each other if one forgets about the time stamp. åTo better support scalability for ontology-based query answering, a technique called "ontology based data access" (OBDA) has been proposed. In ontology-based data access, an ontology with moderate expressivity (DL-Lite [6]) is used to represent the conceptual domain model such that queries (or event specifications) can be transformed (rewritten and unfolded) in a sound and complete way in order to execute them on a conventional database system or triplestore. In addition, OBDA introduces the use of so called mapping rules for mapping database representations into ontology representations, with the advantage that ontology representations need not be materialised since mapping rules are considered in the query rewriting process.

Both, temporalizing OBDA (e.g. [4],[14]) and streamifying OBDA (see, e.g., [18] for RDFS++) are hot topics in current investigations. In particular, state sequences are investigated w.r.t. to OBDA on streaming data such that queries can be specified using formulas from linear temporal logic (LTL) [4]). Compare also the languages used in *complexed event processing*, e.g., EP-SPARQL/ETALIS [2]. Nonetheless, tests with different streaming benchmarks show (see e.g. [19]) that implemented systems for stream processing w.r.t. conceptual data models are just at the beginning of their development.

The plethora of approaches indicates that there is no single way to deal with streaming data. Nonetheless, we argued that the bag-semantics used in the RDFS or DL stream systems from above may lead to problems. Hence, in the next section we will explore an extensible framework based on the ideas of ontology-based data access for scalability and window-based query specification for expressivity and flexibility such that event specifications or specifications in linear temporal logic can be integrated.

2 Sensor Measurement Scenarios

The sensor measurement scenario is considered to be a typical application scenario for stream reasoning, and it can easily be embedded into the context of Linked Open Data and the Semantic Web (see `http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/` for a nearly standard ontology for semantic sensor networks). Hence, we develop our query language with examples from this scenario, thus enabling better comparability with other approaches.

Before going into the details of the scenario we recapitulate the usual use of the stream notion. A *temporal stream* is a set of pairs (d, t). The first argument is instantiated by an object d from a domain D, which we call the domain of streamed objects. The second argument is instantiated by a timestamp t from a structure $(T, \leq)$ of linearly ordered timestamps. The elements of the the stream are thought to be arriving at the query answering system in some order. In the *synchronized stream setting*, which will be in the focus of this article, one demands that the timestamps in the arrival ordering make up a monotonically increasing sequence. In the context of OBDA for streams, we will have to deal with two different domains of streamed objects. The first domain consists of relational tuples from some schema; in this case we call the stream a *relational* stream. The second domain is made up by data facts, either represented as ABox assertions or as RDF triples (and as such may be inhomogeneous).

2.1 A Concrete Example

We discuss a gas turbine monitoring and control application scenario, which is provided by one of the industrial stakeholders (SIEMENS) within the EU funded FP7 project Optique (`http://www.optique-project.eu/`). STARQL is used and tested in this project also within this scenario.

Assume that few service centers are run in order to monitor, analyze, and control hundreds of turbines in various power plants. The storage system in the data center can be viewed as a central DB, which stores different types of information, the most relevant being static data on the one hand, such as turbine infrastructure data, as well as, on the other hand, time-stamped measurement data stemming from many sensors. In addition, so-called event data ("messages") from control units are stored with timestamps indicating the creation time of the events. Control units are small computation units, seen as black boxes, getting input from sensors.

A (normalized) relational DB schema for the monitoring scenario is shown below for demonstration purposes.

```
SENSOR(SID, CID, Sname, TID, description)
SENSORTYPE(TID, Tname)
COMPONENT(CID, superCID, AID, Cname)
ASSEMBLY(AID, AName, ALocation)
MEASUREMENT(MID, MtimeStamp, SID, Mval)
MESSAGE(MesID, MesTimeStamp, MesAssemblyID,
        catID, MesEventText)
CATEGORY(catID, catName)
```

The schema models two categories of messages, measurements and event messages. The latter are produced by control units and are partitioned into categories (`categoryID`). Some messages contain warnings or failure hints, others indicate the operational status of the turbine (start up, running, stopping) and so on. The infrastructure data contains the description of sensors, their names, and their types, the components to which they are attached, and, in turn, the assemblies to which these are attached.

It is well known and holds also for the monitoring use case that continuous queries are useful for online predictive diagnosis. But they can also be effectively used for event detection in a reactive diagnosis setting by "replaying" measurements, i.e., simulating NOW being advanced while starting in the past (maybe faster than in realtime). Different sets of conceptual domain models might be used during these "simulations". Please note that ontology-based query answering is very important in this context because for different simulations one might use different ontologies without changing the queries manually.

2.2 Lifting the Data to the Logical Level

Accessing data trough the interface of an ontology presupposes a method to lift the data stored in a SQL database or arriving in a relational stream into the logical level of ontologies. In the OBDA setting the chosen method is that of mappings, formally realized as rules with logical queries on the left hand and SQL on the right hand.

In the monitoring scenario, assume that the ontology signature contains a concept symbol $Sens$ (for sensors) and an attribute symbol $name$. The following mapping induces the set of ABox assertions stating which individuals are sensors and how they are named.

$Sens(x), name(x, y) \longleftarrow$
```
SELECT f(SID) as x,Sname as y FROM SENSOR
```

The lefthand side of the mapping is a conjunctive query and the right-hand side is a SQL query in which all the variables of the CQ are used. The information in the row of the measurement table is mapped to unary facts ($Sens(x)$) and binary atomic facts ($name(x, y)$). A similar mapping could determine all burner tip temperature sensors $BTTSens$.

A procedure for complete answering queries w.r.t. a TBox, which contains intensional knowledge, demands to incorporate the implicit entailments of the data w.r.t. the TBox. The perfect rewriting approach behind OBDA does not use classical reasoning on the TBox and ABox, but compiles a given query using the TBox axioms into a new query, which is evaluated directly onto the data and yields sound and complete answers. So, the ABox is not materialized.

The axioms in the pure DL-Lite TBox have to be understood as stating that at every time point every burner tip temperature sensor is a temperature sensor, similarly for temperature sensors and sensors.

Besides these traditional mapping, one has to declare mappings with time tags. As an example, we describe a mapping that gives the values y which a sensor x shows at time point z.

$val(x, y)\langle z\rangle \longleftarrow$
```
SELECT f(SID) AS x, Mval as y,
MtimeStamp AS z FROM MEASUREMENT
```

The ABox induced by such mappings is called a *temporal ABox.*

Quite similar to the mappings producing time tagged assertions we can construct mappings from relational streams to streams of time tagged assertions. In the mapping above, one just has to replace the reference to the table MEASUREMENT by a reference to a named relational stream with measurements, say $relS_{Msmt}$ (cf. the stream mapping language S2O in [5]).

To make it concrete, we assume that the relational streams are described in the stream relational query language CQL [3]. So assume that $relS_{Msmt}$ is a stream of time tagged tuples, which arrive over a TCP socket and may, e.g., be produced by a simulation stream on the table MEASUREMENT. The stream mapping then would be

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

The query language which will be introduced in the next section operates on the ontological level, its inputs being TBoxes, static ABoxes, temporal ABoxes and (not necessarily homogeneous) streams of ABox assertions. The stream of ABox assertions underlying most of the following examples is the measurement stream S_{Msmt}. 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 $0s$.

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

3 The Query Language STARQL

Now let us see informally, how an appropriate query language for sensor data streaming scenarios could look like within the challenging paradigm of OBDA. We will describe the query language *STARQL (Streaming and Temporal ontology Access with a Reasoning-based Query Language)* on the abstract logical level, thereby assuming that the data in the databases and the relational streams have already been mapped (virtually) to the logical level of TBoxes, static or temporal ABoxes, and ABox assertion streams.

3.1 A Basic Example

For the ease of exposition let us first assume that the terminological TBox is empty. The engineer may be interested whether the temperature measured in the sensor s_0 grew monotonically in the last two seconds, i.e., in the interval $[NOW - 2s, NOW]$. We first present the solution in our new streaming language STARQL and use it to explain the main conceptual ideas, the most important one being that of *ABox sequencing*.

```
CREATE STREAM S_out AS
SELECT { s0 rdf:type RecentMonInc }<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ
HAVING  FORALL i < j IN SEQ,?x,?y:
 IF ({ s0 val ?x }<i>  AND { s0  val ?y }<j>)
 THEN ?x <= ?y
```

The solution is centered around a window of range 2s. Every 1 second, which is determined by the slide parameter and denoted above by ->1s, the window moves forward in time and gathers all timestamped assertions whose timestamp lies in the interval [NOW-2s, NOW]. Here, NOW denotes the current time point. The development of the time has to be specified locally in the query or directly for all queries by some pulse function.

At every time point, the window content is a set of timestamped assertions which together make up a temporal ABox. For the first two time points $0s$, $1s$ we do not get well defined intervals for $[NOW - 2s, NOW]$, but it is natural to declare the contents at $0s$ and $1s$ as the set of timestamped ABox assertions which have arrived up to second 0 resp. 1. For the other time points, we have proper intervals, and so the resulting temporal ABoxes from $0s$ to $5s$ are defined as follows.

Time	Temporal ABox
$0s$	$\{val(s_0, 90^\circ)\langle 0s\rangle\}$
$1s$	$\{val(s_0, 90^\circ)\langle 0s\rangle, val(s_0, 93^\circ)\langle 1s\rangle\}$
$2s$	$\{val(s_0, 90^\circ)\langle 0s\rangle, val(s_0, 93^\circ)\langle 1s\rangle, val(s_0, 94^\circ)\langle 2s\rangle\}$
$3s$	$\{val(s_0, 93^\circ)\langle 1s\rangle, val(s_0, 94^\circ)\langle 2s\rangle, val(s_0, 92^\circ)\langle 3s\rangle\}$
$4s$	$\{val(s_0, 94^\circ)\langle 2s\rangle, val(s_0, 92^\circ)\langle 3s\rangle, val(s_0, 93^\circ)\langle 4s\rangle\}$
$5s$	$\{val(s_0, 92^\circ)\langle 3s\rangle, val(s_0, 93^\circ)\langle 4s\rangle, val(s_0, 95^\circ)\langle 5s\rangle\}$

Now we have at every second a set of timestamped assertions. In order to apply ABox and TBox reasoning we group these assertions together into (pure) ABoxes. The result of the grouping is a finite sequence of ABoxes (ABoxes are sequenced w.r.t. the order of their timestamps). We can use this sequence of classical ABoxes to filter out those values v for which it is provable that $val(s_0, v)$ "holds".

The ABox sequencing operation is introduced by the keyword `SEQUENCE BY` in the query. There may be different methods to get the sequence, the most natural one is to merge all assertions with the same timestamp into the same ABox. The query above refers to this built-in sequencing method `StdSeq`. Other sequencing methods may be defined by following an SQL-like create declaration.

In our simple example, the standard ABox sequencing leads to simple ABoxes, as the stream does not contain ABox assertions with the same timestamp. Please note that ABoxes in a sequence can be combined with larger static ABoxes (see below). Just for illustration, we note that the ABox sequence at time 5s is defined as follows:

Time	ABox sequence
$5s$	$\{val(s_0, 92°)\}\langle 3s\rangle, \{val(s_0, 93°)\}\langle 4s\rangle, \{val(s_0, 95°)\}\langle 5s\rangle$

The sequence at 5s contains the timestamped ABoxes, e.g., the first one is the ABox $\{val(s_0, 92°)\}$ with timestamp $\langle 3s\rangle$.

Now, as there is at every time point a sequence of ABoxes, one can refer, at every time point, to every (pure) ABox within the sequence in order to apply DL reasoning. This is actually done in the query above within a so-called `HAVING` clause. Here, the `HAVING` clause is a boolean expression. In general, it may be some predicate logical formula with open variables.

The evaluation of the expression `{ s0 val ?x }<i>` (or $val(s0, x)\langle i\rangle$ in DL notation) relies on DL deduction: Find all x that can be proved to be a filler of $value$ for s_0 w.r.t. to the ABox $\mathcal{A}_i$. A TBox and other (static) ABoxes used for deduction can be mentioned in a `USING` clause. Similarly, `{ s0 val ?y }<j>` means that one wants to find all values y which are provably recorded in sensor s_0 w.r.t. the j^{th} ABox in the sequence. The resulting values x, y must be such that the y (recorded in the later ABox j) must be larger than the former.

The result of the monotonicity query is a stream of ABox assertions of the form $RecMonInc(s_0)\langle t\rangle$, defined by the form being specified after the `SELECT` keyword. The timestamp template `<NOW>` indicates that the assertions for the output stream are timestamped with the evolving time parameter of the sliding window. For the first seconds until 5s the output stream is defined as follows:

$$\begin{aligned} S_{out}^{\leq 5s} &= \{RecMonInc(s_0)\langle 0s\rangle, RecMonInc(s_0)\langle 1s\rangle, \\ &\quad RecMonInc(s_0)\langle 2s\rangle, RecMonInc(s_0)\langle 5s\rangle\} \end{aligned}$$

So, the query correctly generates assertions saying that there were recent monotonic increases according to the query for time points 0s, 1s, and 2s, and then again only at time point 5s.

As the monotonicity condition could be useful for other different queries, our language also provides the possibility to define names for `HAVING` predicates. We call these definitions "aggregators". The aggregator definition is demonstrated in the following listing.

```
CREATE AGGREGATE OPERATOR monInc(SEQ,f(*)) AS
  FORALL i < j in SEQ,x,y:
  IF   (f(x)<i>  AND  f(y)<j>)  THEN x <= y
```

The aggregation operator $monInc(\cdot, \cdot)$ gets two arguments, the first being a sequence of ABoxes and the second being a boolean functional in one open argument, i.e., an expression standing for a term with one argument, which if substituted results in a boolean term. In the case of the new query, the functional is given by `?sens val *`. Here, the variable `?sens` is thought to be already bound. The open argument is marked by a star $\star$. Please note that $monInc(\cdot, \cdot)$ is just one example used here for illustration purposes.

3.2 Multiple Streams

Many interesting time series features have to combine values from different streams. In the monitoring scenario, this could be the combination of the measurement streams with the event streams in order to detect correlations. The simplest combination is that of a union of streams which is directly provided in the same comma-separated notation as in pure SQL.

```
SELECT { ?sens rdf:type RecentMonInc }<NOW>
FROM     S_Msmt_1 [NOW-2s, NOW]->1s,
         S_Msmt_2 [NOW-4s, NOW]->2s
 ...
```

But in many cases, the simple union of the ABoxes is not an appropriate means, because the timestamps of the streams might not be synchronized, so the simple idea of joining ABoxes with the same timestamp may lead to an ABox sequence that is well defined but pragmatically irrelevant. For example, if some temperature sensor is recorded to show some value at time 8:35h 12s whereas a different (pressure) sensor shows some value at a slightly different time 8:35h 33s, then the engineer may have an interest in converting these assertions into the same ABox: Because, only if the values of both sensors fulfil some conditions "at the same time" (read as "in the same ABox") will the engineer be able to infer additional knowledge.

For these situations, STARQL provides means for directly defining other (non-standard) sequencing operation, which can be referred to within the `SEQUENCE BY` clause of a query. For example, STARQL provides sequencing constructors based on typical similarity/equivalence relations on timestamps. The similarity relations must obey the time ordering, so that the similarity classes can be considered as focussing-out operations on the timestamps. Having different sequencing constructors means a general modeling flexibility which cannot be simulated conveniently with preprocessing steps or with mappings on the streams. In particular, one can think of sequencing methods for which it is necessary to invoke deduction in the grouping of the ABox in order to check, e.g., consistency.

4 The HAVNG-Clause Sublanguage

The strength of STARQL lies within its high expressiveness of its `HAVING` clause language, which is a first-order logic sub-language allowing also for all-quantifiers, implication, disjunction, negation, and concrete domains such as the real numbers. So, it has to be tamed with safety conditions in order to reach the aim of classical OBDA, which is to reduce queries on the ontological level (here STARQL queries) equivalently to queries on the backend data sources (here: SQL and stream queries such as CQL posed to relational data stream management system). The main observation to guide the construction for the safety conditions of STARQL is that SQL like languages fulfil the property of *domain independence*.

We will roughly sketch the semantics of HAVING clauses after the discussion of another STARQL query in which reasoning over a TBox is needed–thereby demonstrating the demanding aspects of a suitable semantics. The following subsections then discus safety conditions aimed at domain independence and last but not least the result stating domain independence for the safe sub-fragment of HAVING clauses.

4.1 Semantics of **HAVING** Clauses

Suppose that an engineer is interested in a subclass of sensors that grew monotonically in the last seconds, namely the subclass of temperature sensors. The data give more information regarding the sensors; so for example, there are temperature sensors which are attached to some components, such as burner tip temperature sensors. We assume first that the TBox $\mathcal{T}$ contains the axioms $BTTSens \sqsubseteq TSens$, $TSens \sqsubseteq Sens$, and while there are mappings generating a specific ABox assertion $BTTSens(s_0)$, there are no mappings for general temperature sensors $TSens$.

Now, we formulate a query, asking for temperature sensors having grown monotonically in the last 2s. Within the WHERE clause one

```
SELECT {?sens rdf:type RecentMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
USING   STATIC ABOX <http://Astatic>,
        TBOX <http://TBox>
WHERE { ?sens rdf:type TSens }
SEQUENCE BY StdSeq AS SEQ
HAVING  monInc(SEQ, ?sens val *)
```

can specify intra-ABox conditions (unions of conjunctive queries) for generating variable bindings (in SPARQL syntax). Here we use just `?sens rdf:type TSens` asking for temperature sensors.

The evaluation of the condition in the WHERE clause and in the HAVING clause incorporates not only the ABoxes in the generated sequence but also the TBox and the static ABox. The incorporation of these resources is specified in the query with the keyword USING. So, if $\mathcal{A}_i$ is an ABox in the sequence, then the relevant local knowledge base at position i in the sequence w.r.t. which the conditions are evaluated is $KB_i = \mathcal{T} \cup \mathcal{A}_{static} \cup \mathcal{A}_i$. Indeed, in order to identify s_0 in the input stream as a temperature sensor, one has to incorporate, first, the fact from the static ABox stating that s_0 is a burner tip temperature sensor, and second the subsumption from the TBox.

In general, the conditions in the WHERE clause and in the HAVING clause may be more complex. In fact we consider unions of conjunctive queries which is known to allow for rewritability w.r.t. DL-Lite ontologies. This condition language goes together with any member of the DL-Lite family that guarantees FOL rewritability. In particular, we will look at DL-Lite with concrete domains in order to be able to represent sensor values (and timestamps) [17]. The advantage of the ABox sequencing approach is that we can rely on the certain answer semantics for this embedded condition queries.

The semantics of HAVING clauses rests on the meaning of the indexed atoms, whose meaning in turn are given by the certain answer semantics for the embedded SPARQL queries. The idea is to view the tuples in the certain answer sets as members of a sorted FOL structure $\mathcal{I}_t$. For a detailed treatment of the semantics of STARQL we refer the user to the technical report [15]. Here we want to make the idea of specifying the semantics of the HAVING clause language a bit more concrete. Assume that the sequence of ABoxes at some given time point t is $seq = (\mathcal{A}_1, \ldots, \mathcal{A}_k)$. Then the domain of $\mathcal{I}_t$ consists of the index set $\{1, \ldots, k\}$ as well as the set of individual constants and the set of value constants. Now, if the HAVING clause contains, for example, the time tagged condition query $(val(s_0, x)\langle i\rangle)$ (with embedded UCQ $val(s_0, x)$), then we introduce for it a binary relation symbol R. This symbol is denoted in $\mathcal{I}_t$ by the certain answers of the embedded query extended with the index i: $R^{\mathcal{I}} = \{(a, i) \mid a \in cert(val(s_0, x), KB_i)\}$. Constants are denoted by themselves in $\mathcal{I}_t$. This already fixes a structure $\mathcal{I}_t$ with finite denotations of its relation symbols. Hence we can consider (the active domain of) $\mathcal{I}_t$ as a finite DB and also speak about evaluating a HAVING clause on this DB.

A major step towards showing transformability in the sense of OBDA is to show that a safe fragment of the HAVING-clause language is domain independent [1] over $\mathcal{I}_t$. The idea of domain independence is that for answering a query on a DB (in logical terms: evaluating a query on an FOL structure) one can rely on the so-called *active domain*, the set of constants occurring in the query and the DB. In particular, as queries and DBs are finite, domain independence guarantees that the result sets are finite.

The formal definition of domain dependence [1] assumes a DB or a FOL structure $\mathcal{I}$ and a query. We will mainly talk about the structure $\mathcal{I}$ or, more concretely, about the corresponding minimal Herbrand model $HB(\mathcal{I})$. Let be given a global domain of possible objects Dom from which all constants in $HB(\mathcal{I})$ stem. The *active domain* of a query q over a structure $HB(\mathcal{I})$, denoted $ad(q, \mathcal{I})$ is the set of all constants appearing in q and $HB(\mathcal{I})$. The set of answers for a query w.r.t. a relativized domain $D \subseteq Dom$, denoted $ans_D(q, \mathcal{I})$, is defined by restricting the domains of all quantifiers in q to D. A query is called *domain independent* iff for all $\mathcal{I}$ and for all D_1, D_2 with $ad(q, \mathcal{I}) \subseteq D_1 \cup D_2 \subseteq Dom$ one has $ans_{D_1}(q, \mathcal{I}) = ans_{D_2}(q, \mathcal{I})$. In particular, for domain independent queries the result set w.r.t. the whole global domain Dom is the same as the result set w.r.t. the active domain.

4.2 Safe **HAVING** Clauses

In its general form (as described in [15]), HAVING clauses are not domain independent. For example, in the HAVING clause of the form $y > 3$ with free concrete domain variable y the result set would be an infinite set of bindings for y, namely, all real number bigger than 3. So we have to describe a safety mechanism which extends the grammar rules for HAVING clauses by adornments for variables, in which the safety status of each variable is mentioned. In the extended grammar, safety adornments are stated first for atomic HAVING clauses and then inductively defined for complex HAVING clauses.

We use markings/adornments for the variables from the set $\{+, -, --\}$, the intuitive meaning being that variables with a tag $+$ are guarded and so can be free in the HAVING clause, variables with tag $-$ are not guarded, but would be guarded if the formula in which they occur were negated, and variables tagged $--$ are not guarded. Then we define exactly those HAVING clauses to be safe that can be constructed with the extended grammar and have only free variables that are tagged with $+$.

Atomic HAVING clauses are either atoms for state indices (such as `i < j` in the monotonicity example above) or value comparisons (such as `?x <= ?y`) or, most importantly, state indexed UCQs such as `{ s0 val ?x }<i>`. Variables in the atoms for state indices are all marked with $+$, the same holds for the indexed UCQs. Value atoms are marked with $--$. For example, we have the following HAVING clauses with their adornments: $i < j(i^+, j^+)$; $x \leq y(x^{--}, y^{--})$; and $val(s0, x) < i > (x^+, i^+)$.

In the inductive construction of the HAVING clauses, the grammar rules update the guard tags of the variables, depending on the logical constructor. For example, consider the grammar rule for constructing an auxiliary HAVING clause $auxhCl$ using an all quantifier.

$$
\begin{array}{l}
auxhCl(\vec{x}_1^{-}, \vec{x}_2^{-}, \vec{y}_1^{-}, \vec{y}_2^{-}, \vec{z}_1^{-}, \vec{z}_2^{--}\vec{j}^{+}, i^{+}) \longrightarrow \\
\texttt{FORALL } \vec{y}' \texttt{ IF } auxhClCQ(\vec{x}_1^{+}, \vec{y}'^{+}, y, \vec{y}_1^{+}, \vec{z}_1^{+}, i^{+}) \texttt{ THEN} \\
auxhCl(\vec{x}_1^{+}, \vec{x}_2^{+}, \vec{y}'^{g}, \vec{y}_1^{-}, \vec{y}_2^{-}, \vec{z}_1^{--}, \vec{z}_2^{--}\vec{j}^{+}, \vec{j}, i^{+}\ )
\end{array}
$$

The rule ensures that variables that are occurring after the implication and that are bounded by the all quantifier are guarded by variables in a conjunction of time indexed atoms (named $auxhClCQ$ above). The remaining variables are updated according to the effects of implication. Other rules in the induction step are constructed in a similar way. (We refer the readers to the complete grammar in the extended version of this paper at `http://www.sts.tu-harburg.de/people/oezcep/papers/papers.html`.)

We give some examples for complex (non-)guarded HAVING clauses. In $val(s0, y)\langle i\rangle \wedge y > 3(y^{+}, i^{+})$, the variable y is safe; we have the conjuncts $val(s0, y)(y^{+})$ and $y > 3(y^{--})$. The grammar rule for conjuncts says that the safer guard (here +) wins over the non-safe guard $--$, so that y is adorned with $+$ in the whole formula. In $\neg(hasVal(s0, y)\langle i\rangle \wedge y > 3)\ (y^{-}, i^{+})$ the variable y is unsafe; but it contrast to the y in clause $y > 3$ it is guarded by an atom, so that a second negation for this formula would make it safe.

4.3 Safe **HAVING** Clauses are Domain Independent

Domain independence is well known to hold for relational algebra. This fact can be used to show that also formulas that are in Safe Range Normal Form (SRNF) and that are range restricted are domain independent (cf. [1, p.86]). HAVING clauses, that are safe in our sense, can be transformed equivalently into SRNF formulas that are indeed range restricted. The reduction gives the following theorem. (For a proof see the extended version).

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

As there is a concrete construction from safe range SRNF formula into relational algebra [1, p.89], one can explicitly (with some minor restrictions on STARQL) construct transformations to CQL and also to ADP [12], a distributed data management system.

5 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 [15]). 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.

ACKNOWLEDGEMENTS

This work has been partially supported by the European Commission as part of the FP7 project Optique (`http://www.optique-project.eu/`).

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[2] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic, 'Stream reasoning and complex event processing in ETALIS', *Semantic Web*, **3**(4), 397–407, (2012).

[3] A. Arasu, S. Babu, and J. Widom, 'The CQL continuous query language: semantic foundations and query execution', *The VLDB Journal*, **15**, 121–142, (2006).

[4] F. Baader, S. Borgwardt, and M. Lippmann, 'Temporalizing ontology-based data access', in *CADE-13*, (2013).

[5] J.-P. Calbimonte, O. Corcho, and A.J.G Gray., 'Enabling ontology-based access to streaming data sources', in *Proceedings of the 9th international semantic web conference on the semantic web - Volume Part I*, ISWC'10, pp. 96–111, Berlin, Heidelberg, (2010). Springer-Verlag.

[6] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodríguez-Muro, and R. Rosati, 'Ontologies and databases: The DL-Lite approach', in *Semantic Technologies for Informations Systems – 5th Int. Reasoning Web Summer School (RW 2009)*, volume 5689 of *LNCS*, 255–356, Springer, (2009).

[7] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M.A. Shah, 'TelegraphCQ: Continuous dataflow processing for an uncertain world', in *CIDR*, (2003).

[8] V. Haarslev and R. Möller, 'Incremental query answering for implementing document retrieval services', in *Proceedings of the International Workshop on Description Logics (DL-2003), Rome, Italy, September 5-7*, pp. 85–94, (2003).

[9] R.J. Hendley, R. Beale, C.P. Bowers, C. Georgousopoulos, C. Vassiliou, P. Sergios, R. Moeller, E. Karstens, and D. Spiliotopoulos, 'CASAM: collaborative human-machine annotation of multimedia', *Multimedia Tools and Applications Journal*, 1–32, (2013).

[10] J.-H. Hwang, Y. Xing, U. Çetintemel, and S. B. Zdonik, 'A cooperative, self-configuring high-availability solution for stream processing', in *ICDE*, pp. 176–185, (2007).

[11] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, and S. Zdonik, 'Towards a streaming SQL standard', *Proc. VLDB Endow.*, **1**(2), 1379–1390, (2008).

[12] H. Kllapi, E. Sitaridi, M.M. Tsangaris, and Y. E. Ioannidis, 'Schedule optimization for data processing flows on the cloud', in *SIGMOD Conference*, pp. 289–300, (2011).

[13] J. Krämer and B. Seeger, 'Semantics and implementation of continuous sliding window queries over data streams', *ACM Trans. Database Syst.*, **34**(1), 1–49, (April 2009).

[14] F. Lécué and J.Z. Pan, 'Predicting knowledge in an ontology stream', in *IJCAI*, ed., F. Rossi. IJCAI/AAAI, (2013).

[15] Ö. L. Özçep, R. Möller, C. Neuenstadt, D. Zheleznyakov, and E. Kharlamov, 'Deliverable D5.1 – a semantics for temporal and stream-based query answering in an OBDA context', Deliverable FP7-318338, EU, (October 2013).

[16] D.L. Phuoc, H.Q. Nguyen-Mau, J.X. Parreira, and M. Hauswirth, 'A middleware framework for scalable management of linked streams', *J. Web Sem.*, **16**, 42–51, (2012).

[17] O. Savkovic and D. Calvanese, 'Introducing datatypes in dl-lite', in *Proc. of the 20th European Conf. on Artificial Intelligence (ECAI 2012)*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pp. 720–725. IOS Press, (2012).

[18] E. Della Valle, S. Ceri, D. Barbieri, D. Braga, and A. Campi, 'A first step towards stream reasoning', in *Future Internet – FIS 2008*, volume 5468 of *LNCS*, 72–81, Springer Berlin / Heidelberg, (2009).

[19] Y. Zhang, P. Minh Duc, O. Corcho, and J.P. Calbimonte, 'SRBench: A Streaming RDF/SPARQL Benchmark', in *Proceedings of International Semantic Web Conference 2012*, (November 2012).

Appendix A: Grammar for STARQL HAVING clauses

We give the grammar for the `HAVING` clause language with its safety mechanism. We let X stand for all variables in the `WHERE` clauses. clause. These variables can be handled as if they were constants, as the instantiations are fixed. So every variable from X occurring in the `HAVING` clause is guarded.

The `HAVING` clause opens a context on which aggregation and filter conditions on a sequence of ABoxes are formulated. The ABox sequence seq is always finite (as the streams are isomorphic to the natural numbers), hence there is a finite interval $I(seq)$ of natural numbers with the usual ordering, which represents the order of timestamps within the window. We loosely use $i \in seq$ instead of $i \in I(seq)$. Regarding the indices we assume the existence of the existence of the first element, denoted by the constant 0, existence of 1 (if at least two elements are in I). Moreover, we assume the existence of a maximum constant, denoted $max(I)$ or just max if the context is clear. Moreover, let be given a plus-relation defined for all $i, j, k \in I$ as the usual addition relation $I \models plus(i, j, k)$ iff $i + j = k$.

Let $\Psi(\vec{x}, \vec{y})$ denote a UCQ with free (distinguished) variable sets $\vec{x}, \vec{y}$. The set of variables $\vec{x}$ are variables among the set of variables $\vec{X}$ used after the `SELECT` keyword and bounded to constants within the `WHERE` clause. Hence, these variables all can be thought of as non-free variables - and so will be handled as such. The set of variables $\vec{y}$ are variables to be used in the `HAVING` clause as distinguished variables or bounded by the existentials.

There are different types of atoms allowed in the `HAVING` clause. The first group is made up by embedded filter queries and an index of the sequence. The second group contains atoms over the concrete domain values. The third group are arithmetic atoms over the indexes. Moreover, there are atoms using macro predicates. We deviate a little bit from the context free representation by allowing for co-occurrences of the same variables on the lefthand side and the righthand side of a grammar rule. An atom $havingAtom(\vec{y_1}^{+}, \vec{y_1}^{-})$ is one containing the variables in $\vec{y_1}^{+}, \vec{y_2}^{-}$, the ones in $\vec{y}_1$ are guarded the other not guarded. Let g_i be a meta variables standing for the guard status of a variable x_i^{g}, $g_i \in \{+, -, --, \}$. We assume the following ordering on the guards:

$$\emptyset \preceq -- \preceq - \preceq +$$

The special case of $g = \emptyset$ is a convenience notation meaning for $x^{\emptyset}$ that x does not occur at all in the formula. The status of all the variables within the following clauses are governed by rules as described in the Figure 5. The entry for $l_1 \wedge l_2$ in row $l_1 = --, l_2 = -$ is $--$. This is to be read as follows: Take any formula $F_1(z^{--}, \vec{x}) \wedge F_2(z^{-}, \vec{y})$, where the subformula F_1 has a variable z which is labelled $--$ and perhaps other variables $\vec{x}$ and where the subformula F_2 contains the same variable z which is labelled$--$ and perhaps other variables $\vec{y}$. Then the marking of the whole formula w.r.t z is $--$. The only restriction is that neither F_1 nor F_2 is an identity assertion.

The rules (for $\wedge, \vee, \rightarrow$) handle only the cases of variable labels co-occurring in the sub-formulae. So the following grammar rules for the `HAVING` clauses have to incorporate also the cases of variables that occur only in one sub-formula but not both. So we extend the table above by the rules in Fig. 5. Now $\emptyset$ for l_i stands for the fact that there is no corresponding variable in formula F_i

As said before, we do not make the completion explicit but only simulate the consequences for the labelings directly on the grammar.

l_1	l_2	$\neg l_1$	$l_1 \wedge l_2$	$l_1 \vee l_2$	$l_1 \rightarrow l_2$
$--$	$--$	$--$	$--$	$--$	$--$
$--$	$-$	$--$	$--$	$-$	$-$
$--$	$+$	$--$	$+$	$--$	$--$
$-$	$--$	$+$	$--$	$-$	$--$
$-$	$-$	$+$	$-$	$-$	$-$
$-$	$+$	$+$	$+$	$-$	$+$
$+$	$--$	$-$	$+$	$--$	$-$
$+$	$-$	$-$	$+$	$-$	$-$
$+$	$+$	$-$	$+$	$+$	$-$

Figure 1. Combination of Guards

l_1	l_2	$l_1 \wedge l_2$	$l_1 \vee l_2$	$l_1 \rightarrow l_2$
$--$	$\emptyset$	$--$	$--$	$--$
$-$	$\emptyset$	$-$	$-$	$--$
$+$	$\emptyset$	$+$	$--$	$-$
$\emptyset$	$--$	$--$	$--$	$--$
$\emptyset$	$-$	$-$	$-$	$-$
$\emptyset$	$+$	$+$	$--$	$--$

Figure 2. Guard combination with non-existing variable in second component

The *havingClause* filter conditions are built on the basis of the *havingAtoms* as FOL formulas using them as subformulae. Here we use the abbreviation *auxhCl* for *AuxiliaryhavingClause*, *hCL* for *havingClause* (= safe auiliary having clause) and *hIndAt* for *hIndexedAtom*. For a vector of variables $\vec{x}$ let $set(\vec{x})$ denote the underlying set of variables. Then for any $op \in \{\cup, \cap, \Delta, \setminus\}$ let : $\vec{x}\ op\ \vec{y} = set(x)\ op\ set(\vec{y})$.

Please note that the guard conditions are of syntactical nature so that one cannot guarantee that the guards are the same for logically equivalent formulae. Take e.g.

$$(\neg(hasVal(s_0, z)\langle i\rangle \wedge z > 4) \wedge \exists j.j \neq j)(z^{-}) \quad (1)$$

This formula is equivalent to bottom (because the second conjunct is) — and bottom has no free variable at all. We can only guarantee the following: For all formula F that are equivalent to a formula $auxhCL$ generated by the grammar it holds that every variable in F has a weaker (or equal) guard than the variable in $auxhCL$.

Appendix B: Proof of Theorem 1

A desirable property of query languages over DBs or more logically, relational FOL structures $\mathcal{I}$ is the independence of the answer set from the chosen domain in the struct. We will talk of mainly of of the structure $\mathcal{I}$ or more concretely of the corresponding minimal herbrand model $HB(\mathcal{I})$. This can be formalized with the notion of domain independence as explicated in the classical textbook on the foundation of databases [1]. Let be given a global domain of possible objects Dom from which all constants in $HB(\mathcal{I})$ stem from. In our case the Dom consists of individual constants and value constants and time index constants. The *active domain* of a query q over a structure $HB(\mathcal{I})$, denoted $ad(q, \mathcal{I})$ is the set of all constants appearing in q and appearing in $HB(\mathcal{I})$. The set of answers of a query

$$
\begin{array}{rcl}
\mathit{indTerm}(i^{+}) & \longrightarrow & i \\
\mathit{indTerm}() & \longrightarrow & \texttt{max} \mid \texttt{0} \mid \texttt{1} \\
\mathit{havingIndexedAtom}(\vec{x}^{\emptyset}, \vec{y}^{+}, i^{+}) & \longrightarrow & \Psi(\vec{x}, \vec{y}) \;\; \texttt{<i>} \\
 & & \text{(for } \Psi(\vec{x}, \vec{y}) \text{ being a UCQ and } \vec{x} \in X, \vec{y} \notin X) \\
\mathit{havingIndexedAtom}(x^{--}, y^{--}) & \longrightarrow & x = y \\
 & & \text{for } y, x \notin X \cup Var_{val}) \\
\mathit{havingIndexedAtom}(x^{+}) & \longrightarrow & x = a \mid a = x \\
 & & \text{(for } a \in (X \cap Var_{ind}) \cup Const_{const}, x \in Var_{ind} \setminus X) \\
\mathit{havingValueAtom}(z_1^{+}) & \longrightarrow & z_1 = v \mid v = z_1 \\
 & & \text{(for } z_1 \in Var_{val} \setminus X, \text{ and } v \in Const_{val}) \\
\mathit{havingValueAtom}(z_1^{+}) & \longrightarrow & z_1 = z_2 \mid z_2 = z_1 \\
 & & \text{(for } z_1 \in Var_{val} \setminus X, \text{ and } z_2 \in X \cap Var_{val}) \\
\mathit{havingValueAtom}(z_1^{--}, z_2^{--}) & \longrightarrow & z_1 = z_2 \\
 & & \text{(for } z_1, z_2 \in Var_{val} \setminus X \\
 & & \text{(for } \mathit{op} \in \{\texttt{<}, \texttt{<=}, \;\texttt{=}, \;\texttt{>}, \;\texttt{>=}\}) \\
\mathit{havingValueAtom}(z_1^{--}, z_2^{--}) & \longrightarrow & z_1 \; \mathit{op} \; z_2 \\
 & & \text{(for } \mathit{op} \in \{\texttt{<}, \texttt{<=}, \;\texttt{>}, \;\texttt{>=}\} \text{ and } z_1, z_2 \in Var_{val} \setminus X \\
\mathit{havingValueAtom}(z_1^{--}) & \longrightarrow & z_1 \; \mathit{op} \; v \\
 & & (\mathit{op} \in \{\texttt{<}, \texttt{<=}, \;\texttt{>}, \;\texttt{>=}\} \text{ and } z_1 \in Var_{val} \setminus X, v \in Val_{const} \\
\mathit{havingValueAtom}(z_1^{--}) & \longrightarrow & z_1 \; \mathit{op} \; z_2 \\
 & & (\mathit{op} \in \{\texttt{<}, \texttt{<=}, \;\texttt{>}, \;\texttt{>=}\} \text{ and } z_1 \in Var_{val} \setminus X, z_2 \in (X \cap Val_{var}) \\
\mathit{havingIndexArithAtom}(i_1^{g_1}, i_2^{g_2}) & \longrightarrow & \mathit{indTerm}_1(i_1^{g_1}) \; \mathit{op} \; \mathit{indTerm}_2(i_2^{g_2}) \\
 & & \text{(for } \mathit{op} \in \{\texttt{<}, \texttt{<=}, \;\texttt{=}, \;\texttt{>}, \;\texttt{>=}\}) \\
\mathit{havingIndexArithAtom}(i_1^{g_1}, i_2^{g_2}, i_3^{g_3}) & \longrightarrow & \texttt{plus(}\mathit{indTerm}_1(i_1^{g_1}), \mathit{indTerm}_2(i_2^{g_2}), \mathit{indTerm}_3(i_3^{g_3})\texttt{)} \\
\mathit{arithAggr} & \longrightarrow & \texttt{COUNT} \;\mid \texttt{AVG} \;\mid \texttt{SUM} \;\mid \texttt{MIN} \;\mid \texttt{MAX}
\end{array}
$$

Figure 3. Grammar Rules for Atomic `HAVING` clauses

$$
\begin{array}{rcl}
auxhCl(\vec{z}^{+}, i^{+}) & \longrightarrow & havingIndAt(\vec{z}^{+}, i^{+}) \\
auxhCl(\vec{z}^{g}) & \longrightarrow & havingIndAt(\vec{z}^{g}) \\
auxhCl(\vec{z}_1^{g_1}, \vec{z}_2^{g_2}) & \longrightarrow & havingIndAt(\vec{z_1}^{g_1}, z_2^{g_2}) \\
auxhCl(z_1^{g_1}, z_2^{g_2}) & \longrightarrow & havingValueAtom(z_1^{g_1}, z_2^{g_2}) \\
auxhCl(i_1^{+}, i_2^{+}) & \longrightarrow & havingIndexArithAtom(i_1^{+}, i_2^{+}) \\
auxhCl(i_1^{+}, i_2^{+}, i_3^{+}) & \longrightarrow & havingIndexArithAtom(i_1^{+}, i_2^{+}, i_3^{+}) \\
auxhClCQ(\vec{z}^{+}, i^{+}) & \longrightarrow & havingIndAt(\vec{z}^{+}, i^{+}) \\
auxhClCQ(\vec{z}_1^{+}, \vec{z}_2^{+}, {i^{+}}_1, {i^{+}}_2) & \longrightarrow & auxcClCQ(\vec{z}_1^{+}, {i^{+}}_1) \wedge auxcClCQ(\vec{z}_2^{+}, {i^{+}}_2)
\end{array}
$$

$$
\begin{array}{rcl}
auxhCl(& & \\
(\vec{x}_1 \cap \vec{x}_2)^{+}, (\vec{x}_1 \setminus (\vec{x}_2 \cup \vec{y}_2 \cup \vec{z}_2))^{-}, & & \\
(\vec{x}_1 \cap \vec{y}_2)^{-}, (\vec{x}_1 \cap \vec{z}_2)^{--}, & & \\
\vec{y}_1^{-}, (\vec{z}_1 \cap \vec{y}_2)^{-}, (\vec{z}_1 \setminus \vec{y}_2)^{--}, \vec{y}_2^{-}, & & \\
(\vec{x}_2 \setminus (\vec{x}_1 \cup \vec{y}_1 \cup \vec{z}_1))^{-}, (\vec{x}_2 \cap \vec{y}_1)^{-}, & & \\
(\vec{x}_2 \cap \vec{z}_1)^{--}, (\vec{z}_2 \cap \vec{y}_1)^{-}, & & \\
(\vec{z}_2 \setminus \vec{y}_1)^{--}, (\vec{z}_2 \setminus \vec{y}_1)^{--}, \vec{i}_1^{+}, \vec{i}_2^{+} & & \\
) & \longrightarrow & auxhCl(\vec{x}_1^{+}, \vec{y}_1^{-}, \vec{z}_1^{--}, \vec{i}_1^{+}) \ \texttt{OR} \\
 & & auxhCl(\vec{x}_2^{+}, \vec{y}_2^{-}, \vec{z}_2^{--} \vec{i}_2^{+})
\end{array}
$$

$$
\begin{array}{rcl}
auxhCl(& & \\
(\vec{x}_1 \cup \vec{x}_2)^{+}, & & \\
(\vec{y}_1 \setminus (\vec{z}_2 \cup \vec{x}_2))^{-}, (\vec{y}_1 \cap \vec{z}_2)^{--}, & & \\
(\vec{z}_1 \setminus \vec{x}_2)^{--} & & \\
(\vec{y}_2 \setminus (\vec{z}_1 \cup \vec{x}_1))^{-}, (\vec{y}_2 \cap \vec{z}_1)^{--}, & & \\
(\vec{z}_2 \setminus \vec{x}_1)^{--} & & \\
) & \longrightarrow & auxhCl(\vec{x}_1^{+}, \vec{y}_1^{-}, \vec{z}_1^{--}, \vec{i}_1^{+}) \ \texttt{AND} \\
 & & auxhCl(\vec{x}_2^{+}, \vec{y}_2^{-}, \vec{z}_2^{--} \vec{i}_2^{+}) \\
 & & \text{where both conjuncts are different from an} \\
 & & \text{identity of the form } x = y \text{ for } x, y \in Var_{ind} \cup Var_{val} \\
auxhCl(z_1^{max\{g_1,g_2,h_1,h_2\}}, z_2^{max\{g_1,g_2,h_1,h_2\}}, \vec{z_3}^{g_3}) & \longrightarrow & auxhCl(z_1^{g_1}, z_2^{g_2}, \vec{z_3}^{g_3}) \ \texttt{AND} \ z_1^{h_1} = z_2^{h_2} \\
auxhCl(\vec{x}^{-}, \vec{y}^{+}, \vec{z}^{--}, \vec{i}^{+}) & \longrightarrow & \texttt{NOT} \ auxhCl(\vec{x}^{+}, \vec{y}^{-}, \vec{z}^{--}, \vec{i}^{+}) \\
\text{Same variable markings as for} & & \\
\texttt{NOT} \ auxhCl(\vec{x}_1^{+}, \vec{y}_1^{-}, \vec{z}_1^{--}, \vec{i}_1^{+}) \ \texttt{OR} & & \\
auxhCl(\vec{x}_2^{+}, \vec{y}_2^{-}, \vec{z}_2^{--} \vec{i}_2^{+}) & \longrightarrow & \ \texttt{IF} \ auxhCl(\vec{x}_1^{+}, \vec{y}_1^{-}, \vec{i}_1^{+}) \ \texttt{THEN} \ auxhCl(\vec{x}_2^{+}, \vec{y}_2^{-}, \vec{i}_2^{+}) \\
auxhCl(\vec{x}^{+}, \vec{y}^{-}, \vec{z}^{--}, \vec{i}^{+}) & \longrightarrow & \texttt{FORALL} \ i' \ \texttt{IN} \ seq \ auxhCl(\vec{x}^{+}, \vec{y}^{-}, \vec{z}^{--}, \vec{i}^{+}, i'^{+}) \mid \\
 & & \texttt{EXISTS} \ i' \ \texttt{IN} \ seq \ auxhCl(\vec{x}^{+}, \vec{y}^{-}, \vec{z}^{--}, \vec{i}^{+}, i'^{+}) \mid \\
auxhCl(\vec{x}_1^{-}, \vec{x}_2^{-}, \vec{y}_1^{-}, \vec{y}_2^{-}, \vec{z}_1^{-}, \vec{z}_2^{--} \vec{j}^{+}, i^{+}) & \longrightarrow & \texttt{FORALL} \ \vec{y}' \ \texttt{IF} \ auxhClCQ(\vec{x}_1^{+}, \vec{y'}^{+}, y, \vec{y}_1^{+}, \vec{z}_1^{+}, i^{+}) \ \texttt{THEN} \\
 & & auxhCl(\vec{x}_1^{+}, \vec{x}_2^{+}, \vec{y}'^{g}, \vec{y}_1^{-}, \vec{y}_2^{-}, \vec{z}_1^{--}, \vec{z}_2^{--} \vec{j}^{+}, \vec{j}, i^{+} \ ) \\
auxhCl(\vec{x}_1^{+}, \vec{x}_2^{+}, \vec{y}_1^{+}, \vec{y}_2^{-}, \vec{z}_1^{+}, \vec{z}_2^{--} \vec{j}^{+}, \vec{j}^{+}, i^{+}) & \longrightarrow & \texttt{EXISTS} \ \vec{y}' hIndAt(\vec{x}_1^{+}, y'^{+}, \vec{y}_1^{+}, \vec{z}_1^{+}, i^{+}) \ \texttt{AND} \\
 & & auxhCl(\vec{x}_1^{+}, \vec{x}_2^{+}, \vec{y}'^{g}, \vec{y}_1^{-}, \vec{y}_2^{-}, \vec{z}_1^{--}, \vec{z}_2^{--} \vec{j}^{+}, i^{+} \ ) \\
hCl(\vec{z}) & \longrightarrow & auxhCl(\vec{z}^{+}) \\
 & & \text{for } z \in Var_{val} \cup Var_{ind}
\end{array}
$$

Figure 4. The non-atoci rules for the `HAVING` grammar

w.r.t. to a relativized domain D with $ad(q, \mathcal{I}) \subseteq Dom$ is defined by restricting the domains of all quantifiers in q to $ans_D(q, \mathcal{I})$, evaluating this query against $\mathcal{I}$ as usual and restricting the tuples such that all components are in D. Now, a query is called *domain independent* iff for all $\mathcal{I}$ and for all D_1, D_2 with $ad(q, \mathcal{I}) \subseteq D_1 \cup D_2 \subseteq Dom$ one has $ans_{D_1}(q, \mathcal{I}) = ans_{D_2}(q, \mathcal{I})$. In particualr, for domain independent queries the answers w.r.t. to the whole global domain Dom is the same as the answer w.r.t. to the active domain.

We will show how to transform HAVING clauses to the relational algebra (SQL). In particular this will show that the HAVING clause language is domain independent as relational algebra is domain independent. The first idea for doing the transformation is to normalize the `HAVING` clause using rules that are also used in generating a formula in *safe range normal form (SRNF)* [1, S.85]. For a formula F let $SRNF(F)$ be the formula resulting from applying the rules in Fig. 5 (until no rule cannot be applied anymore). A formula F is said to be in SRNF iff $F = SRNF(F)$.

1. Rename variables such that no variable symbol occurrence is bound by different quantifiers and such that no variable occurs bound and free
2. Eliminate occurrence of $F \rightarrow G$ by substituting with occurrences of $\neg F \vee G$.
3. Eliminate double negations
4. Eliminate $\forall$ quantifiers by $\forall z \rightsquigarrow \neg \exists z \neg$.
5. Push $\neg$ through using de Morgan rules;

Figure 5. Normalization rules for HAVING clause

Domain independence for formulas in SRNF are handled in the literature citeabiteboul95foundations also by a guard concept. This is realized by a function rr see Figure 6 that is simpler than our guard notion as it presumes formulas in SRNF form. (But note that the second and third rules are my adaptations to relational algebra with concrete domains.)

1. $rr(r(t_1, \dots, t_n))$ = variables in $t_1, \dots, t_n$.
2. $rr(x\ op\ y) = \emptyset$ for $x, y \in Var_{val}, op \in \{<, >\}$
3. $rr(x\ op\ v) = rr(x\ op\ v) = \emptyset$ for $x \in Var_{val}, v \in Const_{val}, op \in \{<, >\}$
4. $rr(x = a) = rr(a = x) = \{x\}$ (for
5. $rr(F \wedge G) = rr(F) \cup rr(G)$
6. $rr(F \wedge (x = y)) = \begin{cases} rr(F) \cup \{x, y\} & \text{if } rr(F) \cap \{x, y\} \neq \emptyset \\ rr(F) & \text{else} \end{cases}$
7. $rr(F \vee G) = rr(F) \cap rr(G)$
8. $rr(\neg F) = \emptyset$
9. $rr(\exists \vec{x} F) = \begin{cases} rr(F) \setminus set(\vec{x}) & \text{if } set(\vec{x}) \subseteq rr(\exists \vec{x} F) \\ \text{return } \bot & \text{else} \end{cases}$

Figure 6. Definition of syntactic range restriction rule

A formula F in SRNF is called *range restricted* iff $free(F) = rr(F)$ and no subformula returned $\bot$.

A well known theorem states that range restricted formula in SRNF are exactly as expressive as relational algebra—which is know to be domain independent. Hence it is well known that safe range SRN formulas are domain independent (which in particular means that all sets of answers are finite).

Theorem 2 *Range restricted formula in SRNF form are domain independent.*

We are now going to show that all `HAVING` clauses are domain independent by relating our guards with the guards for of [1], showing that the resulting equivalent formula in SRNF is indeed safe range.

Now we can proof the theorem, which is restated here.

All `HAVING` clauses (considered as queries on the DBs of certain asnwers within an actual ABox sequence) are domain independent.

Let $hcl(\vec{u}^+)$ be a safe `HAVING`. Let $hclNF(\vec{u}) = SNFR(hcl(\vec{u}^+)$ be the formula resulting from applying the normalization rules in Fig. 5. The status of all the guards are not changed by the rules. Now, we see that for all subformula $G(\vec{x}^+, \vec{y}^-, \vec{z}^{--})$ in $hclNF(\vec{x})$ we have

(*) $rr(G) = set(\vec{x}) =$ all positively marked variables in x

The proof of $(*)$ is by structural induction on construction of the formula $hclNF(\vec{x})$. Let $G(\vec{x}^+, \vec{y}^-, \vec{z}^{--})$ be an atomic clause. Then $rr(G) = set(\vec{x})$ follows directly from the definitions. The case of conjunction is clear too as any $+$ guard combines with any other guard to $+$. Now take negation $G = \neg F(\vec{x}^+, \vec{y}^-, \vec{z}^{--})$. The definition of rr for the negation case says $rr(G) = \emptyset$. Actually we know that F is an atomic formula. Looking at all guards for these formulas in the grammar we see that no one of these is marked with $-$, hence actually $set(\vec{y}) = \emptyset$ and we have $G(\vec{x}^-, \vec{z}^{--})$, so there is no positively marked variable in G, hence indeed we get that $rr(G) = \emptyset =$ the positively marked variables in G. The case for disjunction is clear as a positive label 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 exists quantifier $G = \exists x F(\vec{x}^+, \vec{y}^-, \vec{z}^{--})$. According to induction assumption $rr(F) = set(\vec{x})$. G may result from a transformation of an exists subformula $\exists x at(x^+, \dots) \wedge F'$ in $hcl(\vec{u}^+)$. So the variable x is by definition in the set $\vec{x}$ of positively marked variables in F, hence $rr(G) = rr(F) \setminus \{x\}$. But G does not occur as free variable in G, hence the set of positively marked variables in G is actually $set(\vec{x}) \setminus \{x\}$ which proves the induction claim. Now G may also result from applying somewhere the rule $\forall \equiv \neg \exists \neg$. But again, there is an atom which is guarding the all quantifier so that one gets again a formula of the form $\exists x at(x^+, \dots) \wedge F'$.