Stream Reasoning – Research and Use cases at the Institute of Information Business

Axel Polleres & Javier Fernandez, WU Wien
Disclaimer: **NO concrete work on Stream Reasoning in my group at the moment**, but still on 3 related topics...

1. Dynamic Linked Data & Open Data
2. Efficient RDF Stream Interchange (ERI)
3. SPARQL 1.1 Update & Entailment as a basis for Stream Reasoning Semantics?
Collecting changes in Linked and Open Data:

• Dynamic Linked Data Observatory

• Open Data Portalwatch

• The DBPedia Wayback machine
1.1 Dynamic Linked Data Observatory

Started in DERI since 2012, **weekly snapshots** of crawl of Linked Data... Interesting dataset for investigating LOD dynamics.

http://dyldo.deri.org/

**Publications:**


1.2 Open Data Portal Watch

http://data.wu.ac.at/portalwatch/

- Started in 2014 – similar idea, different dataset:
  - Weekly crawls of Data and Metadata from over 90 Open Data Portals (CKAN, Socrata, etc)
- Goal: Quality assessment
- Evolution tracking
  - Meta data
  - Data

1.2 Open Data Portal Watch

**Available snapshots**

- August: 79080030
- September: 75447664
- October: 7818
- November: 7983
- December: 6000
- January: 81580164228422
- February: 104210423
- March: 13195

**Evolution of quality measures**

- Quality measures over time.
Dbpedia Wayback Machine

http://data.wu.ac.at/wayback/  ... Different concept: create historical RDF Data from wikipedia/dbpedia revision history

"The DBpedia wayback machine" by Javier Fernandez, Patrik Schneider, Jürgen Umbrich, September 2015, SEMANTiCS2015 (Best Poster).
Motivation: Data Integration and System Interoperation at Scale


http://wikistream.wmflabs.org/
1. Dynamic Linked Data & Open Data – Discussion:

Stream Reasoning related questions:

• How can we efficiently store and query archival data?
• Where does reasoning help us? (e.g. detect anomalies, conflicts, assume defaults in case of missing crawls, etc.)

2. Efficient RDF Stream Interchange (ERI)

- **Lightweight Binary RDF (HDT)**
  - Highly *compact* serialization of RDF (slightly bigger than GZIP)
  - Compact RDF store (without prior decompression)
  - Basics for successfully projects
    - Linked Data fragments
    - LOD Laundromat

- **RDF Data Streams**

  ```
  user1.observation [t1]
  weather1.observation [t1]
  user2.observation
  [t3]
  ...
  w1  w2  w3
  u1  u2  u3  u4
  Stream
  ```
In the light of IoT/Industry 4.0, etc. ...

- We are monitoring the W3C RSP (RDF Stream Processing) CG, but also EXI (Efficient XML Interchange and WoT WG)
## Efficient Serialization of streams of LD

<table>
<thead>
<tr>
<th></th>
<th>Plain: Turtle/ Trig/ JSON-LD</th>
<th>Plain + Compression (e.g. gzip)</th>
<th>HDT</th>
<th>Streaming HDT</th>
<th>RDSZ</th>
<th>RDF/XML + EXI</th>
<th>ERI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Streamable</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Scalable</strong></td>
<td>Limited</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Easy (fast) to create and parse</strong></td>
<td>Yes</td>
<td>Limited</td>
<td>Limited</td>
<td>Yes</td>
<td>Limited</td>
<td>Limited</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Compact</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Limited</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Parametrizable:</strong></td>
<td>No</td>
<td>Limited</td>
<td>Yes</td>
<td>No</td>
<td>Limited</td>
<td>Limited</td>
<td>Yes</td>
</tr>
<tr>
<td>compression/time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ERI: Easy to recover information
- HDT: HTTP Deep Trace
- RDSZ: RDFS-Sizer
- RDF/XML + EXI: Recursively Defined Format XML + EXI

Efficient Serialization of streams of LD.
(Assumption) Most RDF streams are well structured
  • the structure is well-known by the data provider
  • the number of variations in the structure are limited

Efficient RDF Interchange (ERI) Format encodes the information at two levels:
  • A sliding dictionary of structures: Structural Dictionary
  • The concrete value for each predicate
Efficient RDF Interchange (ERI) Format – Basic Concepts

Stream

molecule

"7.7"^^xsd:float

"9.4"^^xsd:float

weather: TemperatureObservation

weather: AirTemperature

ssn:observedProperty

ex:CelsiusValue

 rdf:type

Casual user

Anual pass

wind

Structural Dictionary

weather:

TemperatureObservation

rdf:type

AirTemperature

ssn:observedProperty

ex:CelsiusValue

???

…”
2. Efficient RDF Stream Interchange (ERI) – Discussion:

Stream Reasoning related questions:

• Where does Reasoning help us? (e.g. compact representation by stripping off implicit information)

• Vague idea what could be related here:

SPARQL 1.1 can update a stream and do entailment regimes, however...

**Standardization** of SPARQL 1.1 Update, and SPARQL 1.1 Entailment Regimes with triple stores implementing those standards (rewriting- or materialization-based) is silent about their interaction
What's been done already?

- What do off-the-shelf triple stores do?
  - **Entailment** typically handled
    - at (bulk) loading by *materialization*, or
    - at query time by *rewriting*
  - but **not in the context of Updates**.
- no “standard” behavior for **Delete** & **Insert** upon materialized stores.
- interplay of Entailments and Update left out in the SPARQL 1.1 spec.

- What does the literature say?
  Approaches in the literature on updates and RDFS (or also DLs) limited to **atomic update** operations...
  - [Gutierrez et al., ESWC2011] ABox deletions in the context of RDFS
  - [Calvanese et al., ISWC2010] ABox & TBox insertions in the context of DL-Lite (incl. inconsistency repair)
  Also related:
  - Deductive DBs: [Gupta et al., SIGMOD93]: **DRed (delete and re-derive)**, applied by [Kotowski et al.2011] and [Urbani et al. 2013] in the context of RDF/RDFS...
  - KB evolution, Belief revision, etc.: Various works in classical AI and philosophy

Particularly, none of these considers the **interplay** between **DELETE**, **INSERT** based on a joint **WHERE** clause as in SPARQL
Particularly:

- RDF Stream Reasoning/Stream Processing could be likewise viewed as **stream of updates** adding or removing triples, but:

  - What does it meant to remove implicit triples?
  - Is overdeletion of effects (Dred) always the right thing to do?

What about the **interplay** between **DELETE, INSERT** based on a joint **WHERE** clause as in SPARQL Updates?
Our initial thoughts on this problem...

Updating RDFS ABoxes and TBoxes in SPARQL

Albin Ahmeti\textsuperscript{1}, Diego Calvanese\textsuperscript{2}, and Axel Polleres\textsuperscript{3}

\textsuperscript{1} Vienna University of Technology, Favoritenstraße 9, 1040 Vienna, Austria
\textsuperscript{2} Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy
\textsuperscript{3} Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

Abstract. Updates in RDF stores have recently been standardised in the SPARQL 1.1 Update specification. However, computing entailed answers by ontologies is usually treated orthogonally to updates in triple stores. Even the W3C SPARQL 1.1 Update and SPARQL 1.1 Entailment Regimes specifications explicitly exclude a standard behaviour for entailment regimes other than simple entailment in the context of updates. In this paper, we take a first step to close this gap. We define a fragment of SPARQL basic graph patterns corresponding to (the RDFS fragment of) \textit{DL-Lite} and the corresponding SPARQL update language, dealing with updates both of ABox and of TBox statements. We discuss possible semantics along with potential strategies for implementing them. In particular, we treat both, (i) materialised RDF stores, which store all entailed triples explicitly, and (ii) reduced RDF Stores, that is, redundancy-free RDF stores that do not store any RDF triples (corresponding to \textit{DL-Lite} ABox statements) entailed by others already. We have implemented all semantics prototypically on top of an off-the-shelf triple store and present some indications on practical feasibility.
Exploring possible ABox update semantics

- Materialized-preserving semantics
  - $\text{Sem}_{0}^{\text{mat}}$ ... baseline semantics
  - $\text{Sem}_{1a}^{\text{mat}}$
  - $\text{Sem}_{1b}^{\text{mat}}$ \{ inspired by DRed: delete (incl. effects) and re-derive new effects upon inserts \}
  - $\text{Sem}_{2}^{\text{mat}}$ ... delete incl. causes and rewrite upon inserts
**Sem$_0^{mat}$:** Naïve Update followed by re-materialization

### TBOX

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Mother</td>
<td>rdfs:subClassOf</td>
<td>:Parent</td>
</tr>
<tr>
<td>:hasMother</td>
<td>rdfs:subPropertyOf</td>
<td>:hasParent</td>
</tr>
<tr>
<td>:hasMother</td>
<td>rdfs:range</td>
<td>:Mother</td>
</tr>
<tr>
<td>:hasParent</td>
<td>rdfs:domain</td>
<td>:Child</td>
</tr>
<tr>
<td>:hasParent</td>
<td>rdfs:range</td>
<td>:Parent</td>
</tr>
</tbody>
</table>

### DELETE

DELETE { ?X a :Child . }
INSERT { ?Y a :Mother . }
WHERE { ?X :hasParent ?Y . }

### INSERT

DELETE { :marie a :Child . }
INSERT { :maria_t a :Mother . }

?X=::marie  
?Y=::maria_t

### No effect!

### ABOX-materialized

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>:marie</td>
<td>:hasMother</td>
<td>:maria_t</td>
</tr>
<tr>
<td>:marie</td>
<td>:hasParent</td>
<td>:maria_t</td>
</tr>
<tr>
<td>:maria_t</td>
<td>a</td>
<td>:Mother</td>
</tr>
<tr>
<td>:maria_t</td>
<td>a</td>
<td>:Parent</td>
</tr>
</tbody>
</table>

**materialize(G)**

---

Note: The above image contains a combination of natural text and diagrams illustrating the process of naive update followed by re-materialization in the context of semantic web technologies, specifically using the SPARQL query language and RDF data models.
Alternative Materialized-pres. semantics

- \textbf{Sem}^{mat}_{1a}
  - “Delete and rederive”

\[
G_{u(P_d,P_i,P_w)}^{Sem^{mat}_{1a}} = \text{materialize}(T \cup (\mathcal{A} \setminus \text{materialize}(T \cup \mathcal{A}_d)) \cup \mathcal{A}_i)
\]

\[
\mathcal{A}_d = \bigcup_{\theta \in ans(P_w,G)} gr(P_d\theta)
\]

\[
\mathcal{A}_i = \bigcup_{\theta \in ans(P_w,G)} gr(P_i\theta)
\]

1. DELETEs triples \textbf{incl. Effects}
2. INSERT triples
3. Re-materialize
Sem$_{1a}^{\text{mat.}}$: Delete and rederive

### TBOX

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Mother</td>
<td>rdfs:subClassOf</td>
<td>:Parent</td>
</tr>
<tr>
<td>:hasMother</td>
<td>rdfs:subPropertyOf</td>
<td>:hasParent</td>
</tr>
<tr>
<td>:hasMother</td>
<td>rdfs:range</td>
<td>:Mother</td>
</tr>
<tr>
<td>:hasParent</td>
<td>rdfs:domain</td>
<td>:Child</td>
</tr>
<tr>
<td>:hasParent</td>
<td>rdfs:range</td>
<td>:Parent</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### ABOX-materialized

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DELETE { :marie :hasMother :maria_t. }

DELETE { :marie :hasP arent :maria_t .  
  :marie a Child .  
  :maria_t a Mother.  
  :maria_t a Parent.  }

May be viewed quite "radical"
Sem$_{1a}^{mat}$: Delete and rederive

G

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Mother</td>
<td>rdfs:subClassOf</td>
<td>:Parent</td>
</tr>
<tr>
<td>:hasMother</td>
<td>rdfs:subPropertyOf</td>
<td>:hasParent</td>
</tr>
<tr>
<td>:hasMother</td>
<td>rdfs:range</td>
<td>:Mother</td>
</tr>
<tr>
<td>:hasParent</td>
<td>rdfs:domain</td>
<td>:Child</td>
</tr>
<tr>
<td>:hasParent</td>
<td>rdfs:range</td>
<td>:Parent</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TBOX

DELETE { :marie :hasParent :maria_t. }

DELETE { :marie :hasParent :maria_t .
          :marie a Child .
          :maria_t a Parent. }

Again: no effect!

ABOX-materialized

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>:marie</td>
<td>:hasMother</td>
<td>:maria_t</td>
</tr>
<tr>
<td>:maria_t</td>
<td>a</td>
<td>:Mother</td>
</tr>
</tbody>
</table>

materialize(G)
Alternative Materialized-pres. semantics

- \textit{Sem}_{1b}^{mat}
  - Variant of \textit{Sem}_{1a}, that makes a distinction between \textit{explicit} and \textit{implicit} triples.
  - Re-materialization from scratch from $\mathcal{A}'_{expl}$

\[
G_{u(P_d,P_i,P_w)}^{Sem_{1b}^{mat}} = \mathcal{T} \cup \mathcal{A}'_{expl} \cup \mathcal{A}'_{impl}
\]

\[
\mathcal{A}'_{expl} = (\mathcal{A}_{expl} \setminus \mathcal{A}_d) \cup \mathcal{A}_i
\]

\[
\mathcal{A}'_{impl} = \text{materialize}(\mathcal{A}'_{expl} \cup \mathcal{T}) \setminus \mathcal{T}
\]
Sem\textsubscript{1b}^{mat}: Delete and rederive with separating "explicit" and "implicit" ABox

G

<table>
<thead>
<tr>
<th>TBOX</th>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Mother</td>
<td>rdfs:subClassOf</td>
<td>:Parent</td>
<td></td>
</tr>
<tr>
<td>:hasMother</td>
<td>rdfs:subPropertyOf</td>
<td>:hasParent</td>
<td></td>
</tr>
<tr>
<td>:hasMother</td>
<td>rdfs:range</td>
<td>:Mother</td>
<td></td>
</tr>
<tr>
<td>:hasParent</td>
<td>rdfs:domain</td>
<td>:Child</td>
<td></td>
</tr>
<tr>
<td>:hasParent</td>
<td>rdfs:range</td>
<td>:Parent</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ABox\textsubscript{expl}

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>:marie</td>
<td>:hasMother</td>
<td>:maria_t</td>
</tr>
</tbody>
</table>

ABox\textsubscript{impl}

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>:marie</td>
<td>:hasMother</td>
<td>:maria_t</td>
</tr>
<tr>
<td>:marie</td>
<td>a</td>
<td>:Child</td>
</tr>
<tr>
<td>:marie</td>
<td>:hasParent</td>
<td>:maria_t</td>
</tr>
<tr>
<td>:maria_t</td>
<td>a</td>
<td>:Mother</td>
</tr>
<tr>
<td>:maria_t</td>
<td>a</td>
<td>:Parent</td>
</tr>
</tbody>
</table>

Abox'\textsubscript{impl}

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>:marie</td>
<td>:hasMother</td>
<td>:maria_t</td>
</tr>
<tr>
<td>:marie</td>
<td>a</td>
<td>:Child</td>
</tr>
<tr>
<td>:marie</td>
<td>:hasParent</td>
<td>:maria_t</td>
</tr>
<tr>
<td>:maria_t</td>
<td>a</td>
<td>:Mother</td>
</tr>
<tr>
<td>:maria_t</td>
<td>a</td>
<td>:Parent</td>
</tr>
</tbody>
</table>

DELETE \{ :marie :hasParent :maria_t. \}

Again: no effect!

materialize(G)
Alternative Materialized-pres. semantics

- **$Sem^\text{mat}_2$**
  - **Delete** the instantiations of $P_d$ plus all their causes;
  - **Insert** the instantiations of $P_i$ plus all their effects.

\[
G^\text{Sem^mat}_u(P_d, P_i, P_w) = G_u(P_d^\text{caus}, P_i^\text{eff}, \{P_w\}\{P_d^\text{fvars}\})
\]

\[
P_d^\text{fvars} = \{x \text{ a rdfs:Resource.} \mid \text{for each } x \in \text{Var}(P_d^\text{caus}) \setminus \text{Var}(P_d)\}
\]
3. SPARQL 1.1 Update & Entailment as a basis for Stream Reasoning Semantics? – Discussion:

- A first step to close the gap left by the current standards (SPARQL1.1 Update vs. SPARQL1.1 Entailment Regimes)
  
- Seemingly no “one-size fits all” semantics: Particularly, DRed not intuitive in all cases!
  
- Query rewriting may in some cases be more efficient then re-materialization
  
- Non-intuitive corner cases in each possible semantics  
  \[\rightarrow\] depends on use case?
  
Which stream scenarios warrant which semantics?