SPARQL1.1 Updates and Entailment - Why the specification is silent about their interaction

Based on joint work with: Albin Ahmeti, Diego Calvanese, Vadim Savenkov

Axel Polleres
web: http://polleres.net
twitter: @AxelPolleres
Outline

1. RDF & Linked Data – as a Data abstraction layer of Databases published on the Web
2. SPARQL - Data – as an abstraction layer of Data services on the Web
3. Two new features in SPARQL 1.1:
   - Entailment regimes
   - SPARQL Update
4. How do they interact?
   - Possible Semantics & Examples
5. Discussion
From the HTML Web...

- Globally Unique identifiers
- Links between **Documents** (href)
- A common protocol
From the HTML Web…

… towards a Web of (Linked) Data

- Globally Unique identifiers
- Typed Links (=relations) between Entities
- A common protocol

→ a universal graph-based data format...
   (note that e.g. any relational data can be decomposed into such triples)
Any data can be represented and linked using RDF
Linked Data has moved from academia to industry over the last few years...
News for venice italy

Tornado tears through parts of Venice, Italy (VIDEOS)
Washington Post (blog) - 16 hours ago
A rare tornado (or waterspout, when over water) swept over several islands (Lido, Sant'Elena and Sant'Erasmo) off Venice's lagoon earlier ...

Italy putting brakes on excitement
London Free Press - 14 hours ago

Italy could be hit by Spanish contagion
Economic Times - 1 day ago

Venice - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Venice
Venice (Italian: Venezia [vonetsja] ( listen), Venetian: Venesia [veˈnesja] is a city in northeast Italy sited on a group of 118 small islands separated by canals and connected by dike and bridges. It is located in the marshy Venetian Lagoon which stretches along the shoreline between the mouths of the Po and the Piave Rivers. Wikipedia

Venice Vacations, Tourism and Venice Italy Travel Reviews ...
www.tripadvisor.com/Tourism-g187870-Venice_Veneto-Vacations.html
Venice Vacations: With 130000 reviews of Venice, Italy travel resources, TripAdvisor is the source for Venice information.

ItalyGuides.it: Virtual tour of Venice, Italy - travel information and city ...
www.italyguides.it/us/venice_Italy/venice_travel.htm
Venice tourism and travel information: transport, attractions, maps, travel advice, pictures, audio guides, airport information, activities, hotels and more in Venice, ...

Official website of the Municipality of Venice - Comune di Venezia
www.comune.venezia.it/it/it/venezia/appuntamenti.php/L/IT/1
Official website of the Municipality of Venice, Italy. News, information and tools available to citizens and visitors.
The “Semantic Web” promise...

“If HTML and the Web made all the online documents look like one huge book, RDF, schema and inference languages will make all the data in the world look like one huge database”

Tim Berners-Lee, 1999

Try the following in google:
- “Children of Austrian rulers married to French rulers...?”
The necessary information is available in RDF – Dbpedia: Wikipedia as a Data Service...

Wikipedia as a Data Service...

S P O
:marie :hasMother :maria_t
:marie :hasSpouse :louis
:maria_t a :DauphinOfFrance
:louis a
SPARQL: query language for RDF

- SPARQL offers a standard protocol/service interface to data offering services like DBPedia!

```sparql
SELECT ?X ?P ?S WHERE {
    ?P a RulerOfAustria . ?S a :DauphinsOfFrance .
}
```

![Query result table](http://dbpedia.org/sparql?default-graph-uri=&query=SELECT+DISTINCT+%3FX+%3FP+%3FS+WHERE+%7B+?X+:+hasMother+?P.+?X+:+hasSpouse+?S.+?P+a+RulerOfAustria.+?S+a:+DauphinsOfFrance.+7%7D)
SPARQL 1.1

- SPARQL 1.0 (2008): SQL "look-and-feel" for RDF and Linked data

- SPARQL 1.1 (2013): adding demanded features

- Added two new features:
  - Use of implicit information: SPARQL 1.1 Entailment regimes
  - Provide an update interface: SPARQL 1.1 Update
SPARQL1.1 Entailment Regimes:

- Make use of ontological meta-information (RDFS and OWL):

```sparql
SELECT ?X ?P ?S WHERE {
  ?P a RulerOfAustria .  ?S a :RulerOfFrance .
}
```

<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>:hasSpouse</td>
<td>:louis</td>
</tr>
<tr>
<td>:maria_t</td>
<td>rdf:type</td>
<td>:RulerOfAustria</td>
</tr>
<tr>
<td>:louis</td>
<td>rdf:type</td>
<td>:DauphinOfFrance</td>
</tr>
</tbody>
</table>

- `:hasMother` rdfs:subPropertyOf :hasParent
- `:DauphinOfFrance` rdfs:subClassOf :RulerOfFrance
How are Entailment Regimes typically implemented in SPARQL engines?
SPARQL Query answering under (RDFS) Entailment:

\[ \text{materialize}(G) \]
SPARQL Query answering under RDFS Entailment:

```
G

TBOX
S      P            O
:Mother    rdf:type      :Parent
:hasMother rdf:type      :P arent
:hasParent rdf:type      :Child
:hasParent rdf:type      :Parent
...  

S      P O
:marie :hasMother :maria_t
:marie a  :Child   
:marie :hasParent :maria_t
:maria_t a  :Mother  
:maria_t a  :Parent  

[15]

SELECT ?Y WHERE {
  :marie :hasParent ?Y .
} UNION

SELECT ?Y WHERE {
  :marie :hasMother ?Y .
} UNION

SELECT ?Y WHERE {
  :marie :hasFather ?Y .
}

Well known:

\text{ans}(\text{rewrite}(q, T), \text{reduce}(G)) = \text{ans}(q, \text{materialize}(\text{reduce}(G)))

```

Bischof et al., ISWC2014] Schema-Agnostic Query Rewriting in SPARQL 1.1
SPARQL 1.1

- SPARQL 1.0 (2008): SQL "look-and-feel" for RDF and Linked data

- SPARQL 1.1 (2013):
  
  - Added two new features:
    - Use of implicit information: SPARQL 1.1 Entailment regimes
    - Provide an update interface: SPARQL 1.1 Update

Many off-the-shelf engines support at least RDFS in one or the other way
Data Services on the Web allow updates:

```
INSERT { :marie :hasChild :marie_therese_of_France .}
```

```
DELETE { ?X ?P ?O .}
WHERE { ?X :hasParent :maria_t .}
```

```
DELETE { ?X :hasParent :maria_t .}
INSE { ?X :hasMother :maria_t. }
WHERE { ?X :hasParent :maria_t .}
```

Examples:
- Standardization of **SPARQL 1.1 Update**, and **SPARQL 1.1 Entailment Regimes** with triple stores implementing those standards (rewriting- or materialization-based)

- Emerging need for a more systematic approach of dealing with SPARQL 1.1 Update over ABoxes & Tboxes

**This Talk: We have run into different directions, but we haven't reached the goal yet!**

*Nothing endures but change.* - Heraclitus
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
Our initial thoughts on this problem...

Updating RDFS ABoxes and TBoxes in SPARQL

Albin Ahmeti, Diego Calvanese, and Axel Polleres

1 Vienna University of Technology, Favoritenstraße 9, 1040 Vienna, Austria
2 Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy
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) 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 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.
Let's start with a minimizing expectations:

• Discuss possible update semantics in the context of materialized and reduced stores & RDFS:
  – Only ABox updates, TBox fixed
  – Use "minimal" RDFS as TBox language (without axiomatic triples, blank nodes) ... i.e., DL-Lite$_{RDFS}$
  – Restrict on BGPs to only allow ABox Insert/Deletes

• Even in this restricted setting (RDFS) defining a reasonable update semantics under entailments turns out to be challenging!
Exploring possible ABox update semantics for SPARQL+RDFS

• Materialized-preserving semantics
  – $Sem_{0}^{mat}$ ... baseline semantics
  – $Sem_{1a}^{mat}$ } inspired by DRed: delete (incl. effects) and re-derive new effects upon inserts
  – $Sem_{1b}^{mat}$
  – $Sem_{2}^{mat}$ ... delete incl. causes and rewrite upon inserts

• Reduced-preserving semantics
  – $Sem_{0}^{red}$ ... baseline semantics
  – $Sem_{1}^{red}$ ... delete incl. causes followed by reduce
Baseline semantics for materialized stores:

- $Sem_0^{mat}$
  - Naïve Update followed by re-materialization

\[
G_u^{Sem_0^{mat}}(P_d, P_i, P_w) = \text{materialize}(G_u(P_d, P_i, P_w))
\]
**Sem**$_{0\text{mat}}$: Naïve Update followed by re-materialization

**G**

**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>: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>

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

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

materialize(G)

?X=:marie
?Y=:maria_t

No effect!
Alternative Materialized-pres. semantics

• $\text{Sem}^{\text{mat}}_{1a}$
  
  – “(Over-)delete and rederive”

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

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

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

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

DETERM { :marie :hasMother :maria_t. }

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

May be viewed quite "radical"
**Sem**$_{1a}^{\text{mat}}$: 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**

<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>

**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>

**DELETE**

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

**DELETE**

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

**Again**: no effect!
Alternative Materialized-pres.
semantics

• \( Sem_{1b}^{mat} \)
  
  – Variant of \( Sem_{1a} \), that makes a distinction between \textit{explicit} and \textit{implicit} triples.
  
  – Re-materialization from scratch from \( A'_{expl} \)

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

\[
A'_{expl} = (A_{expl} \setminus A_d) \cup A_i
\]

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

DELETE { :marie :hasParent :maria_t. }

Again: no effect!
Alternative Materialized-pres. semantics

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

\[
G_u(Sem_2^{mat}) = G_u(P_{d^{caus}}, P_{i^{eff}}, \{P_w\}\{P_d^{fvars}\})
\]

\[
P_d^{fvars} = \{?x \text{ a rdfs:Resource.} | \text{ for each } ?x \in Var(P_d^{caus}) \setminus Var(P_d)\}
\]
### ABOX

<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>
</tbody>
</table>

### 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>

### ABOX-materialized

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

### G

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

**rewrite**(u,T)

**INSERT** { ?Y a :Mother. ?Y a :Parent. }
**WHERE** { { ?X :hasMother ?Y. }
{ ?x1 a rdfs:Resource. ?x2 a rdfs:Resource. ?x3 a rdfs:Resource. } }

**DELETE** { :marie a :Child. :marie :hasFather :maria_t, :marie, :louis ... .
:marie :hasMother :maria_t, :marie, :louis ... .
:marie :hasParent :maria_t, :marie, :louis ... . }
**INSERT** { :maria_t a :Mother . :maria_t a Parent . }

?X=:marie ?Y=:maria_t
?x1=?x2=?x3=*

**materialize**(G)
DELETE {}
INSERT {{marie :hasMother :maria_t;:hasFather :louis}
WHERE {};

DELETE {{marie :hasMother :maria_t;;:hasFather :louis }
INSERT{}
WHERE{};

DE​LETE {}  
INSERT {{:marie :hasMon​ther :maria_t;:hasFather :louis,  
:marie :hasParent :maria_t, :hasParent :louis​. 
:louis a :Father, Parent.} WHERE {};

DELETE {{marie :hasMother :maria_t;:hasFather :louis }
INSERT {}
WHERE {};

DELETE following INSERT is NOT idempotent!

⇒ Leaves "Dangling effects"
Baseline semantics

- $Sem_0^{red}$
  - Naïve Update followed by re-reduce

$$G_{u(P_d,P_i,P_w)}^{Sem_0^{red}} = red(G_{u(P_d,P_i,P_w)})$$
**Sem$_0^{red}$**: Naïve Update followed by re-reduce

**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>:hasParent</td>
<td>rdfs:range</td>
<td>:Mother</td>
</tr>
<tr>
<td>:hasParent</td>
<td>rdfs:domain</td>
<td>:Child</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ABOX-reduced**

<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>

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

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

?X=:marie
?Y=:maria_t

No effect!
Alternative Reduced-pres. semantics

- **Sem$_{1}^{red}$**
  - **Delete** the instantiations of $P_d$ plus all their causes;
  - **Insert** the instantiations of $P_i$.
  - Followed by **re-reduce**

\[
G_{u}(P_{d}, P_{i}, P_{w}) = \text{red}(G_{u}(P_{d}^{\text{caus}}, P_{i}, \{\text{rewrite}(P_{w})\})\{P_{d}^{\text{fvars}}\})
\]

\[
P_{d}^{\text{fvars}} = \{?x \text{ a rdfs:Resource.} \mid \text{for each } ?x \in \text{Var}(P_{d}^{\text{caus}}) \setminus \text{Var}(P_{d})\}
\]
**Sem$^{red}_{1}$**: Re-written Update followed by re-reduce

---

### G

#### 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-reduced

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

---

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

**rewrite(u,T)**

**INSERT** { ?Y a :Mother. }
**WHERE** { { ?X :hasMother ?Y. }
{ ?x1 a rdfs:Resource. ?x2 a rdfs:Resource.  
 ?x3 a rdfs:Resource.  }

---

**DELETE** { ?X a :Child. :marie :hasMother :maria_t, :louis, ...  
 :marie :hasParent :maria_t, :louis, ...  
 ...  }
**INSERT** { :maria_t a :Mother. }

---

**reduce(G)**

?X=:marie ?  
Y=:maria_t  
?x1=?x2=?x3=*

---

**36**
Extensions

• TBox updates
• (ABox) Updates in the presence of ontologies beyond RDFS: dealing with inconsistencies.
• Experiments & Use cases
  – Bring mappings into play: Ontology-based data management
Extension 1: TBox updates

• In this setting
  – We expand BGPs to take into account TBox Inserts/Deletes – **general BGPs**
  – Tbox **inserts** are trivial in this setting of RDFS.
  – TBox **deletions** for RDFS are ambiguous
    (need *minimal cuts*) [Gutierrez et al., ESWC2011]

• Opens various degrees of freedom... What if we consider a materialized (acyclic) Tbox?
  – We also use two RDFS rules for **TBox materialization**.

```sql
DELETE { :A rdfs:subClassOf ?C . }
```
DELETE { :A rdfs:subClassOf :F . }

DELETE { :B rdfs:subClassOf :C . :B rdfs:subClassOf :D . }

DELETE { :B rdfs:subClassOf :C. : D rdfs:subClassOf :E . }

DELETE { :E rdfs:subClassOf :F }

Minimal cuts are ambiguous!
Proposed TBox update semantics

- Outbound cut: Intuition

DELETE { :A rdfs:subClassOf ?X. }
WHERE
{ :A rdfs:subClassOf ?X .
  ?X rdfs:subClassOf* :F. }

Idea: Can be implemented with SPARQL 1.1 property paths
Analogous alternative: $\text{Sem}_{\text{incut}}$

- Inbound cut: Intuition

Downside of incut/outcut:
- Only works for acyclic Tboxes
- Minimality of cut only guaranteed for single atomic updates
Extension 2: Beyond RDFS

Dealing with Inconsistencies due to Class Disjointness in SPARQL Update

Albin Ahmeti\textsuperscript{1,3}, Diego Calvanese\textsuperscript{2}, Axel Polleres\textsuperscript{3}, and Vadim Savenkov\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. The problem of updating ontologies has received increased attention in recent years. In the approaches proposed so far, either the update language is restricted to (sets of) atomic updates, or, where the full SPARQL Update language is allowed, the TBox language is restricted to RDFS where no inconsistencies can arise. In this paper we discuss directions to overcome these limitations. Starting from a DL-Lite fragment covering RDFS and concept/class disjointness axioms, we define two semantics for SPARQL Update: under cautious semantics, inconsistencies are resolved by rejecting \textit{all} updates potentially introducing conflicts; under brave semantics, instead, conflicts are overridden in favor of new information where possible. The latter approach builds upon existing work on the evolution of DL-Lite knowledge bases, setting it in the context of generic SPARQL updates.

Minimal ontology language extension: RDFS + class disjointness.
SPARQL updates: deal with inconsistency within *new knowledge*

:Monarch owl:disjointWith :Regent
:hasRegent rdfs:domain :Monarch
:hasRegent rdfs:range :Regent

Unsafe update $\Rightarrow$ intrinsically inconsistent:

```
INSERT {?X :hasRegent ?Y} WHERE {?Y :signsDecreeInNameOf ?X}
```
SPARQL updates: deal with inconsistency within *new knowledge*

:Monarch owl:disjointWith :Regent
:hasRegent rdfs:domain :Monarch
:hasRegent rdfs:range :NonRoyalRuler

"Unsafe" update:

INSERT {?X :hasRegent ?Y} WHERE {?Y :signsDecreeInNameOf ?X}

intrinsically Inconsistencies can be caught by "safe rewriting"

INSERT {?X :hasRegent ?Y}
WHERE {?Y :signsDecreeInNameOf ?X}
MINUS { ?X1 :signsDecreeInNameOf ?Y}
UNION {?X :signsDecreeInNameOf ?Y2}}

Copies of the WHERE clause, variables renamed appropriately.
Rewritings: MINUS vs. FILTER NOT EXISTS

*Safe rewriting via FILTER NOT EXISTS doesn't work:*

```
DELETE{?Z a :Regent} INSERT{?X :hasRegent ?Y}
WHERE{  {?Y :signsDecreeInNameOf ?X}
    UNION {?V :signsDecreeInNameOf ?Z}}
```
Rewritings: MINUS vs. FILTER NOT EXISTS

Safe rewriting via FILTER NOT EXISTS doesn't work:

```sparql
DELETE{?Z a :Regent} INSERT{?X :hasRegent ?Y}
WHERE{
  {?Y :signsDecreeInNameOf ?X}
  UNION {?V :signsDecreeInNameOf ?Z}}
FILTER NOT EXISTS{
  {?X1 :signsDecreeInNameOf ?Y} UNION {?V1 :signsDecreeInNameOf ?Z1}
}
FILTER NOT EXISTS{
  {?X :signsDecreeInNameOf ?Y2} UNION {?V2 :signsDecreeInNameOf ?Z2}
}
```

Simply renaming the whole WHERE clause is not possible.
Rewritings: MINUS vs. FILTER NOT EXISTS

Safe rewriting via MINUS: works!

DELETE{?Z a :Regent} INSERT{?X :hasRegent ?Y}
WHERE{ {?Y :signsDecreeInNameOf ?X}
  UNION {?V :signsDecreeInNameOf ?Z}}

MINUS{
  {?X1 :signsDecreeInNameOf ?Y} UNION {?V1 :signsDecreeInNameOf ?Z1}
}
MINUS{
  {?X :signsDecreeInNameOf ?Y2} UNION {?V2 :signsDecreeInNameOf ?Z2}
}

• MINUS removes variable bindings of the WHERE clause *that can be combined with some result of the query in its right-hand side.*
• Only variables from the left-hand side of MINUS are "visible" in ist right-hand side: great for our case!
SPARQL updates: deal with inconsistency w.r.t. the old knowledge

Adapt the \textit{Sem}_2^{mat} (rewriting-based) semantics

- **Brave**: when in conflict, prefer new knowledge (cf. FastEvol [Calvanese et al 2010])

- **Cautious**: when in conflict, stick to the old knowledge
  - In batch updates, allow variable bindings only where the insert clause does not clash.

- **Fainthearted**: relaxation of cautious semantics
  - The same batch update might resolve clashes by deleting conflicting parts of the old knowledge!
Example: Brave $Sem_{2}^{mat}$

:Monarch owl:disjointWith :Regent
:hasRegent rdfs:domain :Monarch
:hasRegent rdfs:range : NonRoyalRuler

INSERT{?X :hasRegent ?Y} WHERE{?Y :signsDecreeInNameOf ?X}

**Preprocess:**

safe rewriting

INSERT{?X :hasRegent ?Y}
WHERE{?Y :signsDecreeInNameOf ?X}
MINUS{ {?Y1 :signsDecreeInNameOf ?X}
UNION {?Y :signsDecreeInNameOf ?X2}}
Example: Brave $Sem_2^{mat}$

:Monarch owl:disjointWith :Regent
:hasRegent rdfs:domain :Monarch
:hasRegent rdfs:range : NonRoyalRuler

INSERT{?X :hasRegent ?Y} WHERE {?Y :signsDecreeInNameOf ?X}

DELETE {?X a :Regent . ?X3 :hasRegent ?X .
?Y a :Monarch . ?Y :hasRegent ?Y3 }

INSERT{?X :hasRegent ?Y . ?X a :Monarch . ?Y a :Regent}
WHERE {?Y :signsDecreeInNameOf ?X

MINUS{ {?Y1 :signsDecreeInNameOf ?X}
UNION {?Y :signsDecreeInNameOf ?X2}}

OPTIONAL{?X3 :hasRegent ?X}
OPTIONAL {?Y :hasRegent ?Y3}
Example: Cautious $Sem_2^{mat}$

:\Monarch owl:disjointWith :Regent
:\hasRegent rdfs:domain :Monarch
:\hasRegent rdfs:range : NonRoyalRuler

\text{INSERT}\{?X :\text{hasRegent} ?Y\} \text{WHERE}\{?Y :\text{signsDecreeInNameOf} ?X\}

\text{Cautious $Sem_2^{mat}$}

\text{INSERT}\{?X :\text{hasRegent} ?Y . ?X a :Monarch . ?Y a :Regent\}
\text{WHERE}\{?Y :\text{signsDecreeInNameOf} ?X\}
\text{MINUS}\{?Y1 :\text{signsDecreeInNameOf} ?X\}
\text{UNION}\{?Y :\text{signsDecreeInNameOf} ?X2\}
\text{MINUS}\{\{?X a :\text{Regent}\} \text{UNION} \{?Y a :\text{Monarch}\}\}\}

\text{Do inserts only with non-clashing variable bindings (assuming materialized store!)}
Example: Fainthearted $Sem_2^{mat}$

:Monarch owl:disjointWith :Regent
:hasRegent rdfs:domain :Monarch
:hasRegent rdfs:range : NonRoyalRuler

DELETE{?X :Regent}
INSERT{?X :hasRegent ?Y} WHERE{?Y :signsDecreeInNameOf ?X}

handled by delete

MINUS{ {?X1 :signsDecreeInNameOf ?X}
  UNION {?Y :signsDecreeInNameOf ?Y2}}
MINUS {{?X a :Regent} UNION {?Y a :Monarch}}}
Fainthearted semantics: pitfalls, e.g. clashes removed by **different** bindings

DELETE {?Z a :Regent}
INSERT {?X :hasRegent ?Y}
WHERE { ... }

- Atomic updates: for each variable binding $\mu$ of the WHERE clause either **both** delete and **insert** or **none**.
- Insert with $\mu_1$ depends on the deletion with $\mu_2$.
- By atomicity, $\mu_1$ also causes insertion (which might depend on the deletion with some $\mu_3$ etc).

Idea: give up update atomicity. Delete for all $\mu_1$ of the WHERE pattern, insert only where not clashing; for this we have to "separate" DELETE and INSERT...
Breaking atomicity by rewriting

DELETE{?X :Regent}
INSERT{?X :hasRegent ?Y}
WHERE{?Y :signsDecreeInNameOf ?X}

DELETE{?X :Regent}
INSERT{?X1 :hasRegent ?Y1}
WHERE{ {?Y :signsDecreeInNameOf ?X}
  OPTIONAL { ... bind ?X1 and ?Y1 "as needed" } }

Details in a forthcoming paper... The rewriting is involved...
Breaking atomicity using transactions

ALTERNATIVE: Split DELETE and INSERT into two queries.

Step 1: Prepare the bindings for insertions and deletions in a temporary graph.

DELETE {?Z a :Regent}
INSERT {?X :hasRegent ?Y}
WHERE {?Y :signsDecreeInNameOf ?X ...}

Step 2: Run insert and delete separately, but in a single transaction.

Step 2a:
DELETE {?Z a :Regent} WHERE {?Z a Regent_del};

Step 2b:
INSERT {?X :hasRegent ?Y} WHERE {?X :hasRegent_ins ?Y};

Modified Cautious rewriting needed (not shown)
Maintaining Wikipedia: Dbpedia & SPARQL Update to the rescue!

Use case idea: Use mappings to update infoboxes and track pages that need updating.
• Import data from infobloxes using declarative mappings.

• Respecting additional taxonomy.
DBPedia Mapping Types:

- **Page title** corresponds to the subject URI.
- **PropertyMapping**: translate infobox property names to RDF predicate names.
- **IntermediateNodeMapping**: create auxiliary resources.
- **ConditionalMapping**: select concept and predicate names based on a condition.
- **GeocoordinateMapping**
- **DateIntervalMapping**: split a “from-to” string.
- **TableMapping** (very rarely used).
Conditional Mapping: Example

```json
{{ ConditionalMapping | cases =
  {{ Condition
    | templateProperty = наставка
    | operator = isSet
    | mapping = {{ TemplateMapping | mapToClass = OfficeHolder | mappings =
      {{ConstantMapping | ontologyProperty = gender | value = http://dbpedia.org/resource/Female}}}}
  }{ Condition
    | operator = otherwise
    | mapping = {{ TemplateMapping | mapToClass = OfficeHolder | mappings =
      {{ConstantMapping | ontologyProperty = gender | value = http://dbpedia.org/resource/Male}}}}
  }
  | defaultMappings =
  {{ PropertyMapping | templateProperty = имя | ontologyProperty = foaf:name }}
  {{ PropertyMapping | templateProperty = роден-дата | ontologyProperty = birthDate }}
  ...
}}
```
Use Case 1: Translating Updates

Questions/Problems:
• Does an edit lead to inconsistencies on other pages?
• Is there a DBpedia update equivalent to this edit?
  If yes: under which semantics?
DELETE {?X :region :Upper_Normandy .  
  ?Y :region :Lower_Normandy .}
INSERT {?X :region :Normandy .  ?Y :region :Normandy}
WHERE
{
  {?X :region :Upper_Normandy .
  }
  UNION
  {?Y :region :Lower_Normandy .
  }
}

DELETE { :Rouen :region :Upper_Normandy .  
  :Le_Havre :region :Upper_Normandy .  ... 
  :Caen :region :Lower_Normandy .  ... 
  :Bayeux :region :Lower_Normandy .  ... }
INSERT { :Rouen :region Normandy .  
  :Le_Havre :region :Normandy .  ... 
  :Caen :region :Normandy .  
  :Bayeux :region :Normandy .  ... }

ON wikiPage=https://en.wikipedia.org/wiki/Caen
UPDATE InfoboxTemplate(French commune).region=NULL
WHERE InfoboxTemplate(French commune).region=[[Lower_Normandy]]

ON wikiPage=https://en.wikipedia.org/wiki/Caen
INSERT InfoboxTemplate(French commune).region=[[Normandy]]
Conclusions

• First steps to close the gap left by the current standards
  (SPARQL1.1 Update vs. SPARQL1.1 Entailment Regimes)

• We looked into materialized-preserving and also reduced-preserving semantics
  – Focus on implementability in off-the-shelf SPARQL engines with "on-board" means
  – Seemingly no “one-size fits all” semantics
  – Even for RDFS only
    → Bad news: adding OWL features makes things probably worse!
      E.g. adding class inconsistency handling adds even more degrees of freedom.
  – Non-intuitive corner cases in each semantics
    → depends on use case?
  – Which Semantics works in practice for which use case?
    • DBpedia/wikipedia updates as "real use case" is only a sketch so far.
  – Which postulates are satisfied by which semantics?
    • Yet Other possible semantics?
  – All our semantics so far are formula-based, can a model-based semantics also be implemented in
    a triple store (e.g. by storing models in different named graphs?)

• Take-home: SPARQL 1.1 Update, i.e. pairing DELETE and INSERT templates with a
  common WHERE clause (BGP matching) imposes a non-trivial challenge!
Prototype & Evaluation

- A prototype in Java using Jena API implementing the proposed update semantics
  - [http://dbai.tuwien.ac.at/user/ahmeti/sparqlupdate-rewriter/index.html](http://dbai.tuwien.ac.at/user/ahmeti/sparqlupdate-rewriter/index.html)

![Update + 14 LUBM Select queries using LUBM(15) dataset](chart.png)