Stream Reasoning Approaches

Emanuele Della Valle
Daniele Dell'Aglio
Alessandro Margara
This work is licensed under the Creative Commons Attribution 3.0 Unported License.

Your are free:

- **to Share** — to copy, distribute and transmit the work
- **to Remix** — to adapt the work

Under the following conditions

- **Attribution** — You must attribute the work by inserting a credits slide stating
  - These slides are partially based on “Tutorial on Stream Reasoning: Managing Velocity and Variety in Big Data at DEBS 2016” by E. Della Valle, D. Dell’Aglio and A. Margara
  - [http://streamreasoning.org/events/srdebs2016](http://streamreasoning.org/events/srdebs2016)

To view a copy of this license, visit [http://creativecommons.org/licenses/by/3.0/](http://creativecommons.org/licenses/by/3.0/)
Agenda

Incremental Materializations of Ontologies
   - DRed
   - DynamiTE
   - TrOWL

Incremental Materialization of Ontologies in Stream Processing
   - IMaRS
   - Sparkwave

Continuous ontology-based query answering
   - ETALIS and EP-SPARQL
   - Stream Reasoning with ASP
   - STARQL

Formal Semantics of Stream Reasoning
   - LARS
Agenda

**Incremental Materializations of Ontologies**
- DRed
- DynamiTE
- TrOWL

**Incremental Materialization of Ontologies in Stream Processing**
- IMaRS
- Sparkwave

**Continuous ontology-based query answering**
- ETALIS and EP-SPARQL
- Stream Reasoning with ASP
- STARQL

**Formal Semantics of Stream Reasoning**
- LARS
Incremental Materialization of Ontologies

Adopt an incremental approach
Compute only the differences that should be removed and added from the materialization
Agenda

Incremental Materializations of Ontologies
- DRed
- DynamiTE
- TrOWL

Incremental Materialization of Ontologies in Stream Processing
- IMaRS
- Sparkwave

Continuous ontology-based query answering
- ETALIS and EP-SPARQL
- Stream Reasoning with ASP
- STARQL

Formal Semantics of Stream Reasoning
- LARS
Overestimation of deletion: Overestimates deletions by computing all direct consequences of a deletion.

Rederivation: Prunes those estimated deletions for which alternative derivations (via some other facts in the program) exist.

Insertion: Adds the new derivations that are consequences of insertions to extensional predicates.
While inserts are not problematic, deletion are difficult to handle. If we delete $p_2$ discusses $p_1$ ($p_2 \rightarrow p_1$), we have

- **overestimate the impact of the deletion** and mark for deletion $p_4 \rightarrow p_1$ that can be derived by $p_4 \rightarrow p_2$ and $p_2 \rightarrow p_1$

- **look for alternative derivation** of $p_4 \rightarrow p_1$ and eventually find the chain $p_4 \rightarrow p_3$ and $p_3 \rightarrow p_1
Incremental Materializations of Ontologies
  – DRed
  – **DynamiTE**
  – TrOWL
Incremental Materialization of Ontologies in Stream Processing
  – IMaRS
  – Sparkwave
Continuous ontology-based query answering
  – ETALIS and EP-SPARQL
  – Stream Reasoning with ASP
  – STARQL
Formal Semantics of Stream Reasoning
  – LARS
Dynamite
Parallel Materialization

Goal:
− Maintain a very dynamic **knowledge base** (i.e. ontology)

Key contributions:
− Parallelized implementation of materialization
− Efficient maintenance of a Knowledge base that changes frequently

Who and when
− Urbani, Margara, Jacobs et al. VUA Amsterdam. 2013-2014

Reference

Code:
− [https://github.com/jrbn/dynamite](https://github.com/jrbn/dynamite)
− Maintenance, activity: unknown
Problem:

- Incrementally maintaining **materialized knowledge base** in the presence of frequent changes

Two types of updates:

- **Addition**: re-computation of the materialization to add new derivations
- **Removal**: deletion of the explicit knowledge, and implicit information no longer valid

Additions: Parallel Datalog semi-naive evaluation.

Removal: two algorithms:

- Classical Dred
- ‘Counting’ variation: does not require a complete scan of the input for every update
  - Only a fragment of RDFS: $\rho_D F$
Dynamite Workflow

Key: Incremental Materialization

Maintain the KB when there are updates.
Dynamite
Incremental Materialization

Load updated triples in into the main memory
Perform semi-naïve evaluation to derive new triples
Add all the new derivations into the B-Tree indices, making them available for querying.

Divide in schema and generic triples
Divide in 3 types of rules
Parallelize: 1 thread per rule
Each triple with a count attribute:

- number of possible rule instantiations that produced t as a direct consequence

For more complex scenarios: iteratively
Agenda

Incremental Materializations of Ontologies
- DRed
- DynamiTE
- **TrOWL**

Incremental Materialization of Ontologies in Stream Processing
- IMaRS
- Sparkwave

Continuous ontology-based query answering
- ETALIS and EP-SPARQL
- Stream Reasoning with ASP
- STARQL

Formal Semantics of Stream Reasoning
- LARS
Goal:
- Maintain a very dynamic expressive ontology (additions and deletions)

Key contributions:
- Efficient maintenance of an OWL2 EL ontology stream that changes frequently
- Optimizations for deletions (targeting performance)
- Approximate Reasoning techniques for targeting OWL2 DL

Who and when:
- Yuan, Pan, University of Aberdeen 2011-2013

Reference

Code:
- http://trowl.eu
- Tutorial support, actively maintained
Ontologies evolve over time!

Adding and removing axioms over time.

**Ontology stream:** sequence of classical ontologies

\[ O(0), O(1), \ldots, O(n) \]

- **Er(i)** axioms to erase from \( O(i) \)
- **Ad(i)** axioms to add into \( O(i) \)

\[ O(i+1) = O(i) \cup Ad(i) \setminus Er(i) \]
Answering queries on snapshots

- Give me all talks interesting for David

New axioms over time
A directed graph:
- Nodes: axioms / entailments
- Edges: derivation relations among axioms / entailments

All entailments are reachable from their justifications
- Easy to identify impacted entailments
TROWL
Delete and re-derive

Erasing:
– Remove all nodes reachable only from the erased axioms
– Removing all corresponding edges

Adding:
– Adding added axioms as new nodes into the graph
– Inferring new results
– Establishing new edges
Stream Reasoning for OWL 2 EL

• Optimizations to reduce the used memory
• Only axioms relevant for classification are stored
  • other results are not stored
Generate a TMS when doing approximation and reasoning

- **Nodes:**
  - Asserted axioms;
  - **Approximated axioms**;
  - Entailed axioms;

- **Edges:**
  - Created during approximation and reasoning
Proposed for dynamic updates on ontologies

Not streaming data processing engine:
- Not dealing with sequences of unbounded triples or graphs
- Stored ontology axioms, mutable ontology over time
- Updates are frequent, not necessarily streaming data (e.g. frequent transactions in RDBMs)

Efficient maintenance of hanging ontologies
- Interesting and expressive language: OWL2 EL
- Approximate rewritings for OWL2 DL
Incremental Materializations of Ontologies
- DRed
- Dynamite
- TrOWL

Incremental Materialization of Ontologies in Stream Processing
- IMaRS
- Sparkwave

Continuous ontology-based query answering
- ETALIS and EP-SPARQL
- Stream Reasoning with ASP
- STARQL

Formal Semantics of Stream Reasoning
- LARS
Materialize the content of a window w.r.t. a Tbox

- Evaluate the query over the materialization
- Increment the number of results

Materialization is hard to be maintained

- Idea: use incremental maintenance techniques to materialize the content of a window
Running Example – Data Model

Instances

1  2  3

Post

discusses

posts

Person
Naïve solution: an example
Naïve solution: an example
Naïve solution: an example
Naïve solution: an example

\[
\begin{align*}
\text{dom}(\text{\textbullet}) & \subseteq \text{\textbullet} \\
\text{rng}(\text{\textbullet}) & \subseteq \text{\textbullet}
\end{align*}
\]
Incremental maintenance: an example

TBOX

\( \text{dom}(\text{ achiever}) \subseteq \) 
\( \text{rng}(\text{achieved}) \subseteq \)

1 \( \in \) 
2 \( \in \)

Della Valle, Dell’Aglio and Margara

POLITECNICO MILANO 1863
Incremental maintenance: an example

TBOX

\[ \text{dom}(\text{\textbullet}) \sqsubseteq \]

\[ \text{rng}(\text{\textbullet}) \sqsubseteq \]

\[ 1 \in \]

\[ 2 \in \]

\[ 3 \in \]

\[ \text{dom}(\text{\textbullet}) \subseteq \]

\[ \text{rng}(\text{\textbullet}) \subseteq \]

\[ \text{TBOX} \]
Incremental maintenance: an example

TBOX

\[ \text{dom}( \text{1} ) \subseteq \] 
\[ \text{rng}( \text{1} ) \subseteq \] 

\text{To be deleted}

1 \[ \in \] 

TBOX

To be deleted

1 \[ \in \] 

\[ \text{dom}( \text{1} ) \subseteq \] 
\[ \text{rng}( \text{1} ) \subseteq \]
Incremental maintenance: an example
Incremental maintenance: an example

TBOX

$\text{dom( }) \sqsubseteq$

$\text{rng( }) \sqsubseteq$

To be renewed

To be added

To be deleted
Agenda

Incremental Materializations of Ontologies
- DRed
- DynamiTE
- TrOWL

Incremental Materialization of Ontologies in Stream Processing
- IMaRS
- Sparkwave

Continuous ontology-based query answering
- ETALIS and EP-SPARQL
- Stream Reasoning with ASP
- STARQL

Formal Semantics of Stream Reasoning
- LARS
Add reasoning in window-based RSPs
  - RDFS (with transitive property)

Time-based windows slide:
  - The materialisation is executed every time the window updates
  - Only part of data changes at each window update
  - Materialisation is (usually) an expensive task

Intuition: the sliding window operator allows to determine when statements will be removed
Variation of DRed for RDF streams

It pushes the maintenance algorithm in the window operator

An IMaRS window is a sliding window with four parameters:

– \( \omega \): the size of the window
– \( \beta \): the slide of the window
– \( T \): the TBox that describes the data model
– \( M \): the maintenance program

One of the central IMaRS concepts is the expiration time
Expiration time

Every time a statement is added to the window, it is annotated with an expiration time.

The expiration time indicates when the statement should be removed from the materialization.
Every time a statement is added to the window, it is annotated with an expiration time.

The expiration time indicates when the statement should be removed from the materialization.
Expiration time

Every time a statement is added to the window, it is annotated with an expiration time.

The expiration time indicates when the statement should be removed from the materialization.

The statement will exit at 13.

The inferred statements will exit at 13.
Every time a statement is added to the window, it is annotated with an expiration time.

The expiration time indicates when the statement should be removed from the materialization.
Expiration time

Every time a statement is added to the window, it is annotated with an expiration time.

The expiration time indicates when the statement should be removed from the materialization.

The inferred statements expire.
Expiration time

Every time a statement is added to the window, it is annotated with an expiration time.

The expiration time indicates when the statement should be removed from the materialization.

The inferred statements expire.
At each window update

- The computation is done through the execution of a **maintenance program**
Expiration time generation

At each window update

- The computation is done through the execution of a **maintenance program**
The computation is done through the execution of a **maintenance program**.
IMaRS
At a glance

DRed
Delete
Rederive
Insert

IMaRS
Lookup
Insert + Renew
The maintenance program computes the delta sets $\Delta^-$ and $\Delta^+$

- It is a **logic program**

The program is executed every time the content changes

- In our context, the program is executed every time the window slides

The program is composed of **maintenance rules**

- A maintenance rule adds a statement in a set (**context**) if the preconditions are satisfied
IMaRS

Generation of the maintenance program

Rewriting functions

Maintenance program generator

Ontological language

Maintenance program

TBox

IMaRS Window $({\omega, \beta})$
Example: DRed

TBOX
tr(\(\mathcal{G}\))

Current time
11

Window (3,1)

Current time: 11

Della Valle, Dell’Aglio and Margara
Example: DRed

TBOX
\( \text{tr}(\text{ }) \)

Current time
12

Window (3,1)

Della Valle, Dell’Aglio and Margara
Example: DRed

TBOX

\[ \text{tr}(\text{window}(3,1)) \]

Current time

13

Window (3,1)

1

2

3

4

10

11

12

Della Valle, Dell’Aglio and Margara
Example: DRed

```
TBOX
\text{tr}(\text{\textbullet})
```

```
Current time
14
```

```
Window (3,1)
```

```
\text{\textbullet} \quad \longleftarrow \quad \text{\textbullet} \quad \longleftarrow \quad \text{\textbullet} \quad \longleftarrow \quad \text{\textbullet} 
```

```
10 \quad 11 \quad 12 \quad 13
```

```
Delete
```

```
S
```

```
1 \quad 2 \quad 3 \quad 4
```

```
\text{\textbullet} \quad \longleftarrow \quad \text{\textbullet} \quad \longleftarrow \quad \text{\textbullet} \quad \longleftarrow \quad \text{\textbullet}
```

```
Della Valle, Dell’Aglio and Margara
```

```
POLITECNICO MILANO 1863
```
Example: DRed

TBOX

\[ \text{tr}(\text{tr}) \]

Current time

14

Window (3,1)

Delete

1

2

3

4

Current time

10 11 12 13
Example: DRed

TBOX
\( \text{tr} \)

Current time
14

Window (3,1)

Delete
Rederive

Della Valle, Dell’Aglio and Margara

POLITECNICO MILANO 1863
Example: IMaRS

**TBOXX**
\[ \text{tr}(\text{S}) \]

**Current time**
11
Example: IMaRS

TBOX

\( tr( ) \)

Current time

12

Window (3,1)

3 1

14

2 1

14

3

2

3

4

1

15

15
Example: IMaRS

TBOX
\[ tr(\text{...}) \]

Current time
13

Window (3,1)

1 4

2 1 15

3 2 15

4 3

Della Valle, Dell’Aglio and Margara
Example: IMaRS

TBOX
\( \text{tr}(\text{Window (3,1)}) \)

Current time
13

Window (3,1)

1 3

2 1

1 3 2

4 4 1

Della Valle, Dell’Aglio and Margara

POLITECNICO MILANO 1863
Example: IMaRS

TBOX
\( \text{tr}(\Box) \)

Current time
13

Window (3,1)

1 3

2 1 3

4 2 1 3

14 15 15 16
Example: IMaRS

TBOX

\[ \text{tr}(\text{tr}()) \]

Current time

14

Window (3,1)

3 ↪ 3  

1 ↪ 1  

2 ↪ 2  

4 ↪ 4  

15 15 16

10 11 12 13
Incremental Materializations of Ontologies
  - DRed
  - DynamiTE
  - TrOWL

Incremental Materialization of Ontologies in Stream Processing
  - IMaRS
  - **Sparkwave**

Continuous ontology-based query answering
  - ETALIS and EP-SPARQL
  - Stream Reasoning with ASP
  - STARQL

Formal Semantics of Stream Reasoning
  - LARS
Goal:
- RDF data stream processing with additional RDF Schema-based entailments (including inverse and symmetric properties).

Key contributions:
- Usage of RETE for stream processing and reasoning, and extension to account for temporal requirements (time windows) and RDF Schema (+inverse and symmetric) entailments

Who and When
- STI Innsbruck (http://sparkwave.sti2.at/), 2011-2013

References

Code
- https://github.com/Rogger/Sparkwave/
- Maintenance, activity: unknown
Sparkwave
Basic principles: the RETE algorithm

We will illustrate how Sparkwave works with the following basic SPARQL query:

- SELECT ?x ?y WHERE
  
  \{ ?x a b .
  ?x c ?y .
  ?y m n
  \}

- We will show it from now on as the following conjunctive query:
  
  - (\(?x a b\) \^ (\(?x c ?y\) \^ (\(?y m n\))

Traditional RETE networks are based on:

- \(\alpha\)-network, to account for intra-pattern conditions
  
  - One node created for each constant in the triple pattern, so as to filter incoming triples (e.g., five nodes in our sample query)

- \(\beta\)-network, to account for inter-pattern conditions
  
  - Partial matches are stored in the network as tokens.
Let's consider the query: $(?x \ a \ b) \land (?x \ c \ ?y) \land (?y \ m \ n)$
Sparkwave additions

– The ε-network generates triples obtained from RDF Schema entailments
Sparkwave adds to RETE...

Sparkwave additions
- The $\varepsilon$-network generates triples obtained from RDF Schema entailments.

<table>
<thead>
<tr>
<th>Rule name</th>
<th>If</th>
<th>Then add</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf1</td>
<td>$(x \ p \ y)$</td>
<td>$(p \ rdf\text{_type} \ rdf\text{_Property})$</td>
</tr>
<tr>
<td>rdfs2</td>
<td>$(p \ rdfs\text{_domain} \ c) \ (x \ p \ y)$</td>
<td>$(x \ rdf\text{_type} \ c)$</td>
</tr>
<tr>
<td>rdfs3</td>
<td>$(p \ rdfs\text{_range} \ c) \ (x \ p \ y)$</td>
<td>$(y \ rdf\text{_type} \ c)$</td>
</tr>
<tr>
<td>rdfs4a</td>
<td>$(x \ p \ y)$</td>
<td>$(x \ rdf\text{_type} \ rdfs\text{_Resource})$</td>
</tr>
<tr>
<td>rdfs4b</td>
<td>$(x \ p \ y)$</td>
<td>$(y \ rdf\text{_type} \ rdfs\text{_Resource})$</td>
</tr>
<tr>
<td>rdfs5</td>
<td>$(p \ rdfs\text{_subPropertyOf} \ a) \ (q \ rdfs\text{_subPropertyOf} \ r)$</td>
<td>$(p \ rdfs\text{_subPropertyOf} \ r)$</td>
</tr>
<tr>
<td>rdfs6</td>
<td>$(p \ rdf\text{_type} \ rdf\text{_Property})$</td>
<td>$(p \ rdfs\text{_subPropertyOf} \ p)$</td>
</tr>
<tr>
<td>rdfs7</td>
<td>$(p \ rdfs\text{_subPropertyOf} \ a) \ (x \ p \ y)$</td>
<td>$(x \ q \ y)$</td>
</tr>
<tr>
<td>rdfs8</td>
<td>$(c \ rdf\text{_type} \ rdfs\text{_Class})$</td>
<td>$(c \ rdfs\text{_subClassOf} \ rdfs\text{_Resource})$</td>
</tr>
<tr>
<td>rdfs9</td>
<td>$(c \ rdfs\text{_subClassOf} \ d) \ (x \ rdf\text{_type} \ c)$</td>
<td>$(x \ rdf\text{_type} \ d)$</td>
</tr>
<tr>
<td>rdfs10</td>
<td>$(c \ rdf\text{_type} \ rdfs\text{_Class})$</td>
<td>$(c \ rdfs\text{_subClassOf} \ c)$</td>
</tr>
<tr>
<td>rdfs11</td>
<td>$(c \ rdfs\text{_subClassOf} \ d) \ (d \ rdfs\text{_subClassOf} \ e)$</td>
<td>$(c \ rdfs\text{_subClassOf} \ e)$</td>
</tr>
<tr>
<td>rdfs12</td>
<td>$(p \ rdf\text{_type} \ rdfs\text{_ContainerMembershipProperty})$</td>
<td>$(p \ rdfs\text{_subPropertyOf} \ rdfs\text{_member})$</td>
</tr>
<tr>
<td>rdfs13</td>
<td>$(x \ rdf\text{_type} \ rdfs\text{_Datatype})$</td>
<td>$(x \ rdfs\text{_subClassOf} \ rdfs\text{_Literal})$</td>
</tr>
</tbody>
</table>

Table 2: Extra entailment rules from OWL

<table>
<thead>
<tr>
<th>Rule name</th>
<th>If</th>
<th>Then add</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv1</td>
<td>$(p \ owl\text{_inverseOf} \ q)$</td>
<td>$(q \ owl\text{_inverseOf} \ p)$</td>
</tr>
<tr>
<td>inv2</td>
<td>$(p \ owl\text{_inverseOf} \ q) \ (x \ p \ y)$</td>
<td>$(y \ q \ x)$</td>
</tr>
<tr>
<td>sym</td>
<td>$(p \ rdf\text{_type} \ owl\text{_SymmetricProperty}) \ (x \ p \ y)$</td>
<td>$(y \ p \ x)$</td>
</tr>
</tbody>
</table>
Sparkwave operates over a fixed schema
- The ε-network is created at pre-processing time.

Limitations
- Expressiveness in the data schema (only RDF Schema + inverse and symmetric properties)
- Background knowledge cannot be too large, as it is incorporated in memory
Incremental Materializations of Ontologies
  - DRed
  - DynamiTE
  - TrOWL

Incremental Materialization of Ontologies in Stream Processing
  - IMaRS
  - Sparkwave

Continuous ontology-based query answering
  - ETALIS and EP-SPARQL
  - Stream Reasoning with ASP
  - STARQL

Formal Semantics of Stream Reasoning
  - LARS
Goal:
  – Logic-based Complex Event Processing and Stream

Key contributions:
  – Modeling of Complex Event Processing and Continuous RDFS reasoning in Prolog
  – Modeling of iterative (recursive) patterns
  – The engine runs on many Prolog systems: SWI, XSB, ...

Who and When:

References

Code:
  – https://code.google.com/p/etalis/
  – Tutorial support
Iterative (recursive) patterns

- An **output** (complex) event is treated as an **input** event of the same CEP processing agent;

A rule-based approach

- **Rules** can **express** complex relationships between events by matching certain **temporal, relational or causal** conditions
- It can specify and evaluate contextual knowledge
ETALIS Language Syntax

ETALIS Language for Events is formally defined by:

\[ P ::= \text{pr}(t_1, \ldots, t_n) \mid P \text{ WHERE } t \mid q \mid (P).q \]

\[ \mid P \text{ BIN } P \mid \text{ NOT}(P).[P, P] \]

- \( \text{pr} \) – a predicate name with arity \( n \);
- \( t_i \) – denote terms;
- \( t \) – is a term of type boolean;
- \( q \) – is a nonnegative rational number;
- \( \text{BIN} \) – is one of the binary operators: SEQ, AND, PAR, OR, EQUALS, MEETS, STARTS, or FINISHES.

Event rule is defined as a formula of the following shape

\[ \text{pr}(t_1, \ldots, t_n) \leftarrow p \]

where \( p \) is an event pattern containing all variables occurring in \( \text{pr}(t_1, \ldots, t_n) \)
Basics

- SPARQL extension (as with other previously seen languages)
- Interval-based: 2 timestamps

RDF stream – a set of *triple occurrences* \(<\langle s, p, o \rangle, t_\alpha, t_\omega \rangle\) where \(<s, p, o\>) is an RDF triple and \(t_\alpha, t_\omega\) are the start and end of the interval.

Operators

- FILTER, AND, UNION, OPTIONAL, SEQ, EQUALS, OPTIONALSEQ, and EQUALSOPTIONAL
  - Be careful with the management of timestamps (see next)
  - E.g.,

\[
\text{AND} - \text{joins } \langle \mu, t_\alpha, t_\omega \rangle \text{ and } \langle \mu', t'_\alpha, t'_\omega \rangle. \text{ The joined tuple has timestamp } t''_\alpha = \min(t_\alpha, t'_\alpha), \ t''_\omega = \max(t_\omega, t'_\omega); 
\]

Special functions: getDuration(), getStartTime(), getEndTime()
Sequence operators and CEP world

- **SEQ**: joins $e_{ti,tf}$ and $e'_{ti',tf'}$ if $e'$ occurs after $e$
- **EQUALS**: joins $e_{ti,tf}$ and $e'_{ti',tf'}$ if they occur simultaneously
- **OPTIONALSEQ, OPTIONALEQUALS**: Optional join variants

**EP-SPARQL**
- **Extended SPARQL interface to ETALIS**

**Sequence**
- $S$
- $e_1$
- $e_2$
- $e_3$

**Simultaneous**
- $e_4$

**EP-SPARQL query**
- **translator**
- **Prolog engine**
- **continuous results**
Continuously search for companies having a larger than 20% stock price increase in less than 15 days without having acquired another company during that period.

```
SELECT ?company WHERE
  { ?company hasStockprice ?price1 }  
SEQ { { ?company hasAcquired ?othercompany } 
OPTIONALSEQ
  { ?company hasStockPrice ?price2 } } }
FILTER (?price2 > ?price1 * 1.2 &&
!BOUND(?othercompany) &&
getDURATION() < "P15D"^^xsd:duration)
```
Agenda

Incremental Materializations of Ontologies
- DRed
- DynamiTE
- TrOWL

Incremental Materialization of Ontologies in Stream Processing
- IMaRS
- Sparkwave

Continuous ontology-based query answering
- ETALIS and EP-SPARQL
- Stream Reasoning with ASP
- STARQL

Formal Semantics of Stream Reasoning
- LARS
Answer Set Programming

Declarative problem solving approach
- “what is the problem?” vs “how to solve the problem?”
- Problem is modeled using a logic program (set of logic rules)
- Correct interpretations (or Answer Sets) correspond to problem solutions

ASP combines:
- Rich yet simple modeling language
  - Negation, disjunction, integrity constraints, weak constraints, aggregates, ...
- High-performance solving capabilities
  - Based on guess/check/optimize strategy
  - Relies on CWA

ASP has its roots in:
- Deductive databases, Logic programming (with negation), KR and NMR, Constraint solving (mostly SAT)
The StreamRule idea

- 2-tier approach: not all dynamic data streams are relevant for complex reasoning
- Enrich the ability of complex reasoning over data streams
- Keep the solution scalable
- Leverage existing engines from both stream processing and non-monotonic reasoning research areas
The StreamRule idea

in other words...

StreamRule is coupling:

• the linked data stream query processing power of \textit{RSP engines}
• the expressivity and reasoning capabilities of Answer Set Programming with the CLINGO4 stream reasoning solver
• … in a 2-tier approach so that the size of the input is reduced as the reasoning task becomes more computationally intensive.
Limitations

Scalability requires adaptation!
StreamRule: Potentials

Complex reasoning over dynamic streams and their temporal dependencies, makes StreamRule suitable for:

- Dynamic optimal planning/routing
- Spatial reasoning, geofencing, access control, tracking
- Inconsistency checking or constraint-based programming (e.g. configuration, diagnosis)
- ...

BUT needs to be investigated further (among others):

- Window size vs program complexity
- Information flow between the components, more flexible coupling, e.g. to adapt window size
- Parallel, distributed computation (e.g. via STORM/SPARK framework, orchestrated Logic Programs, …)
Incremental Materializations of Ontologies
- DRed
- DynamiTE
- TrOWL

Incremental Materialization of Ontologies in Stream Processing
- IMaRS
- Sparkwave

Continuous ontology-based query answering
- ETALIS and EP-SPARQL
- Stream Reasoning with ASP
- STARQL

Formal Semantics of Stream Reasoning
- LARS
Addressed task:
- Continuous query answering over data streams

Key contributions:
- Use of expressive ontology languages to cope with complex use cases
- (Partially) cover the semantics of temporal ontology languages

Who and When:
- Lubeck University, 2013-ongoing

(Some) Publications:
- ÖL Özçep, R Möller, C Neuenstadt, “A Stream-Temporal Query Language for Ontology Based Data Access”. Description Logics, 2014
Streaming and Temporal ontology Access with a Reasoning-based Query Language

- A framework to access and query heterogeneous sensor data through ontologies

STARQL implementation in an OBDA system:
- An ontology to give an holistic view over the static and streaming data
- Query are composed using the ontology concepts
- Meet the author(s) at the demo session:
  - Enabling Semantic Access to Static and Streaming Distributed Data with Optique, E. Kharlamov et al.
STARQL
A two-layer framework

Example:
- In gas turbine monitoring, detect critical sensors when, in a 5-minute window:
  - There is a monotonic increase of the sensor value for 2 minutes
  - Followed by a failure
STARQL
A two-layer framework

- STARQL is a 2-layer framework
  - STARQL(OL,ECL) composed by:
    - an Ontology Language (OL) to model the data and its schema
    - an Embedded Constraint Language (ECL) to compose the queries
STARQL
A two-layer framework

Examples:

  - FOL-rewritability property
- STARQL(SHI, GCQ): Grounded Conjunctive Queries over SHI ontologies
  - Expressive language for more complex domains
The inputs of a STARQL query are static Tboxes $T^i$, static Aboxes $A^i_{st}$ and streaming ABoxes $S_i$

The syntax of the query is similar to a SPARQL CONSTRUCT query:

```
CONSTRUCT $\Theta_1(x,y)<timeExp_1>,..., \Theta_r(x,y)<timeExp_r>
FROM winExp_1,...,S_m winExp_m,A^0_{st},...,A^k_{st},T^0,...,T^l
WHERE $\psi(x)$
SEQUENCE BY seqMeth
HAVING $\phi(x,y)$
```
The query that detects the critical sensors in STARQL is:

```sql
CREATE STREAM Sout AS
CREATE PULSE AS START = 0s, FREQUENCE = 10s
CONSTRUCT { ?s :a inCriticalState } <NOW>
WHERE { ?s :a TempSens }
SEQUENCE BY StdSeq AS SEQ
HAVING
EXIST i1, i2, i3 in SEQ
  0 < i1 AND i2 < max AND i3 = i2 + 1 AND
  ts(i2) - ts(i1) >= 2min AND
GRAPH i3 { ?s :message ?m . ?m :a A-Message } AND
FORALL i, j in SEQ,?x,?y:
  IF i1 <= i AND i<= j AND j <= i2 AND
  GRAPH i { ?s :val ?x } AND GRAPH j { ?s :val ?y }
THEN ?x <= ?y
```
Agenda

Incremental Materializations of Ontologies
  – DRed
  – DynamiTE
  – TrOWL

Incremental Materialization of Ontologies in Stream Processing
  – IMaRS
  – Sparkwave

Continuous ontology-based query answering
  – ETALIS and EP-SPARQL
  – Stream Reasoning with ASP
  – STARQL

Formal Semantics of Stream Reasoning
  – LARS
Addressed task:
- A high-level **unified** formal foundation for stream reasoning that captures query answering and non-monotonic deduction and enables better comparison/benchmarking

Key contributions:
- a framework to explain and capture the existing Stream Reasoning approaches
- windows as first class citizen in formulas

**Who and When:**
- TU Vienna, 2013-ongoing

Publications:
- Harald Beck, Minh Dao-Tran, Thomas Eiter, Michael Fink: LARS: A Logic-Based Framework for Analyzing Reasoning over Streams. AAAI 2015: 1431-1438H.
- Harald Beck, Minh Dao-Tran, Thomas Eiter: Answer Update for Rule-Based Stream Reasoning. IJCAI 2015: 2741-2747
Formulas

- Formula elements:
  - Window operators ⊞ (substream generation)
  - Boolean connectives: ∧, ∨, →, ¬
  - Temporal/modal operators: ◇, □, @t

- Formulas are defined by the grammar:
  \[
  \alpha ::= a | \lnot \alpha | \alpha \land \alpha | \alpha \lor \alpha | \alpha \rightarrow \alpha | \diamond \alpha | \Box \alpha | \@t \alpha | \Box_i \alpha
  \]

Where:
\( \alpha \): \( \alpha \) holds now
Boolean connectives work as in first order logic
\( \diamond \alpha \): \( \alpha \) holds at some time instant in the past
\( \Box \alpha \): \( \alpha \) holds every time in the past
\( \@t \alpha \): \( \alpha \) holds at the time instant \( t \)
By default, a formula $\alpha$ refers to the whole stream content.

The window $\square_i^\alpha$ is used to set the scope (substream) on which $\alpha$ applies. $\square_i^\alpha$ is a reference to a window function (identified by $i$) that, given a time instant $i$ and a stream, generates a substream with $\pm x$ timestamps from $i$ (by default the counting goes backward, “+” goes forward).

- CQL sliding windows are defined in the framework: Time-based sliding windows, Tuple-based sliding windows and partition-based sliding windows.

Windows can be combined to compose new formulas, e.g. in the last 60 minutes, $\alpha$ holds for 5 (continuous) minutes:

$$\square_i^{60} \diamond \square_i^{5} \alpha$$

(where $\square_i^{60}$ and $\square_i^{5}$ are two time-based sliding windows of 60 and 5 minutes)
Based on datalog-style rules (grounding/solving)
Inherit properties of stable model semantics:
  – Minimality of models
  – supportedness
Each formula in the rule can use operators in the framework
  – Language appears not very intuitive
  – Need some suitable form of program reduct for negation
Offers advanced features:
  – Nondeterminism (multiple choice)
  – Preference and recursion
Can capture:
  – CQL queries (including aggregates and orders)
  – ETALIS operators
Past, current and future work

- Past: lack of theoretical underpinning for stream reasoning
Past, current and future work

- Past: lack of theoretical underpinning for stream reasoning
- Now (April 2015): a (basic) language with precise semantics for
  - Flexible window operator (first class citizen)
  - Time reference/time abstraction
  - Rule-based language for generating intensional data
  - Relationship with other languages (CQL, ETALIS, …)
Past, current and future work

- Past: lack of theoretical underpinning for stream reasoning
- Now (April 2015): a (basic) language with precise semantics for
  - Flexible window operator (first class citizen)
  - Time reference/time abstraction
  - Rule-based language for generating intensional data
  - Relationship with other languages (CQL, ETALIS, …)
- Planned: extended complexity analysis and incremental evaluation (generalizing Truth Maintenance Systems)
Past, current and future work

Past: lack of theoretical underpinning for stream reasoning

Now (April 2015): a (basic) language with precise semantics for

- Flexible window operator (first class citizen)
- Time reference/time abstraction
- Rule-based language for generating intensional data
- Relationship with other languages (CQL, ETALIS, …)

Planned: extended complexity analysis and incremental evaluation (generalizing Truth Maintenance Systems)

Eventually: distributed setting, heterogeneous nodes (Multi-Context Systems)
Stream Reasoning Approaches

Emanuele Della Valle
Daniele Dell'Aglio
Alessandro Margara