Stream Reasoning
For Linked Data
J-P Calbimonte, D. Dell'Aglio,
E. Della Valle, M.I. Ali and A. Mileo
http://streamreasoning.org/events/sr4ld2015

Other Stream Reasoning
Approaches
Jean-Paul Calbimonte, Oscar Corcho, Daniele Dell'Aglio,
Emanuele Della Valle, Alessandra Mileo and Özgür L. Özçep
Share, Remix, Reuse — Legally

- 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 “Streaming Reasoning for Linked Data 2015” by J-P Calbimonte, D. Dell'Aglio, E. Della Valle, M. I. Ali and A. Mileo

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

- Incremental Maintenance Materializations of Ontologies
  - IMaRS
    - done in the previous section
  - Sparkwave
  - DynamiTE: Parallel Materialization of Dynamic RDF Data
  - RDF Stream Reasoning with GPUs
  - Ontology Stream Reasoning with Truth Maintenance Systems

- Continuous ontology-based query answering
  - C-SPARQL/SPARQL\textsubscript{stream}/CQEL Languages
    - done in the previous sessions
  - ETALIS and EP-SPARQL
  - Stream Reasoning with ASP
    - done in the previous section

- Formal Semantics of Stream Reasoning
  - LARS
  - STARQL
Agenda

- Incremental Maintenance Materializations of Ontologies
  - IMaRS
  - done in the previous section
  - Sparkwave
  - DynamiTE: Parallel Materialization of Dynamic RDF Data
  - RDF Stream Reasoning with GPUs
  - Ontology Stream Reasoning with Truth Maintenance Systems

- Continuous ontology-based query answering
  - C-SPARQL/SPARQL\textsubscript{stream}/CQEL Languages
  - done in the previous sessions
  - ETALIS and EP-SPARQL
  - Stream Reasoning with ASP
  - done in the previous section

- Formal Semantics of Stream Reasoning
  - LARS
  - STARQL
Sparkwave

- **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:
    - (\(\text{?x a b} \land \text{?x c ?y} \land \text{?y m n}\))

- 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: $(\exists x \ a \ b) \land (\exists x \ c \ ?y) \land (?y \ m \ n)$
Sparkwave adds to RETE...

- Sparkwave additions
  - The $\varepsilon$-network generates triples obtained from RDF Schema entailments
  - The $\beta$-network nodes check if partial or complete pattern matches apply for the current time window.
Sparkwave adds to RETE...

- Sparkwave additions:
  - The $\varepsilon$-network generates entailments.
  - The $\beta$-network nodes check if partial or complete pattern matches apply for the current time window.

<table>
<thead>
<tr>
<th>Table 1: RDF/RDFS entailment rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule name</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>rdf1</td>
</tr>
<tr>
<td>rdfs2</td>
</tr>
<tr>
<td>rdfs3</td>
</tr>
<tr>
<td>rdfs4a</td>
</tr>
<tr>
<td>rdfs4b</td>
</tr>
<tr>
<td>rdfs5</td>
</tr>
<tr>
<td>rdfs6</td>
</tr>
<tr>
<td>rdfs7</td>
</tr>
<tr>
<td>rdfs8</td>
</tr>
<tr>
<td>rdfs9</td>
</tr>
<tr>
<td>rdfs10</td>
</tr>
<tr>
<td>rdfs11</td>
</tr>
<tr>
<td>rdfs12</td>
</tr>
<tr>
<td>rdfs13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Extra entailment rules from OWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule name</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>inv1</td>
</tr>
<tr>
<td>inv2</td>
</tr>
<tr>
<td>sym</td>
</tr>
</tbody>
</table>
Algorithm 1 Garbage collection

for triple ← epsilon.processedTriples do
  for token ← triple.tokens do
    token.removeTokenFromNode()
  end for
  epsilon.removeTokens(triple)
end for

threshold ← currentTime – timeWindow
for alphaMemory ← rete.alphaMemories do
  for triple ← alphaMemory.triples do
    if triple.timestamp < threshold then
      if triple ∉ staticTriples then
        triple.remove()
      end if
    end if
  end for
end for
- Sparkwave operates over a fixed schema
  - The $\epsilon$-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
Agenda

- Incremental Maintenance Materializations of Ontologies
  - IMaRS
    - done in the previous section
  - Sparkwave
  - **DynamiTE: Parallel Materialization of Dynamic RDF Data**
  - RDF Stream Reasoning with GPUs
  - Ontology Stream Reasoning with Truth Maintenance Systems

- Continuous ontology-based query answering
  - C-SPARQL/SPARQL$_{\text{stream}}$/CQEL Languages
    - done in the previous sessions
  - ETALIS and EP-SPARQL
  - Stream Reasoning with ASP
    - done in the previous section

- Formal Semantics of Stream Reasoning
  - LARS
  - STARQL
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
  - 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: $\rho_{DF}$
- Maintenance of an RDF database

- 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.
Each triple with a count attribute:
- number of possible rule instantiations that produced t as a direct consequence

For more complex scenarios: iteratively
Dynamite

Evaluation: Compare with DRed

- Evaluation with LUBM dataset
  - Classical RDF processing benchmark dataset
  - Not really a streaming dataset

<table>
<thead>
<tr>
<th>Update</th>
<th>Addition (sec.)</th>
<th>Removal (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DRed</td>
<td>Counting</td>
</tr>
<tr>
<td>Update 1</td>
<td>0.117</td>
<td>2902.7</td>
</tr>
<tr>
<td>Update 2</td>
<td>8.2</td>
<td>2049.6</td>
</tr>
<tr>
<td>Update 3</td>
<td>3.7</td>
<td>2121.9</td>
</tr>
<tr>
<td>Update 4</td>
<td>31.8</td>
<td>2132.2</td>
</tr>
<tr>
<td>Update 5</td>
<td>16.8</td>
<td>2196.0</td>
</tr>
<tr>
<td>Update 6</td>
<td>30.5</td>
<td>3830.2</td>
</tr>
</tbody>
</table>

Table 2: Runtime of four type of updates on 138 million triples (LUBM(1000)).

- 1 triple
- 16k triples
- 8k triples
- ~Input size
- 1,2 universities
Dynamite Discussion

- Stored data knowledge base
  - Not a stream of events or facts
  - Traditional RDF database, high number of transactions per time
  - No streaming queries, streaming updates on changes

- Efficient materialization via parallelization techniques

- Multithreaded implementation, optimizations for deletions compared to traditional Dred

- Only a fragment of RDFS
Agenda

- Incremental Maintenance Materializations of Ontologies
  - IMaRS
    - done in the previous section
  - Sparkwave
  - DynamiTE: Parallel Materialization of Dynamic RDF Data
  - **RDF Stream Reasoning with GPUs**
  - Ontology Stream Reasoning with Truth Maintenance Systems

- Continuous ontology-based query answering
  - C-SPARQL/SPARQL\textsubscript{stream}/CQEL Languages
    - done in the previous sessions
  - ETALIS and EP-SPARQL
  - Stream Reasoning with ASP
    - done in the previous section

- Formal Semantics of Stream Reasoning
  - LARS
  - STARQL
RDF Stream Reasoning with GPUs

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

- **Key contributions:**
  - Parallelized implementation of materialization in GPU
  - Efficient maintenance of a Knowledge base that changes frequently

- **Who and when**
  - Liu, Urbani, Qi. VUA Amsterdam, U Maryland, U Southeast China 2014

- **Reference**

- **Code:**
  - No
  - Maintenance, activity: unknown
RDF Stream Reasoning with GPUs

- **Stream (KB,S)**
  - KB: background knowledge (RDF graph)
  - S: stream, sequence of timestamped triples \((\tau, t_i)\)

- **Problem:** For each instant \(t\), decide:
  - RDF graph \(G_t\), such that \(KB \cup S[t-w,t] \vdash G_t\)
  - Given a window \(w\).

- **Correspondence to Temporal RDF**
  - Deductive system, extension of \(\rhoDF\) rules (subset of RDFS)
  - Correspondence of stream at time \(t\):  
    - \(\{\tau:[0,+\infty] | \tau \in KB\} \cup \{\tau:[t',t'+w]|(\tau,t') \in S \text{ and } t'<t\}\)
  - Use Temporal RDF deductive system to compute closure
RDF Stream Reasoning with GPUs

**Implementation**
- GPU CUDA
- RDF graph -> three column table
- Rule execution -> join over tables
- Tbox never changes during streaming

**Workflow**
- Execute transitive closure in Abox in static KB
- When triples arrive:
  - Remove expired triples (out of the window)
  - Compression of RDF stream
  - Parallel Execution of Rules
- Tbox triples cached in GPU memory
- Join between Tbox and incremental part of the ABox
Discussion

- Included concept of stream as input
  - Opposed to previous similar work assuming changes on ontologies
- Includes windowed execution of the stream
- GPU implementation: parallelized code for computing the derivations
- Work in progress, no detailed evaluation and only a short description
Agenda

- **Incremental Maintenance Materializations of Ontologies**
  - IMaRS
    - done in the previous section
  - Sparkwave
  - DynamiTE: Parallel Materialization of Dynamic RDF Data
  - RDF Stream Reasoning with GPUs
  - **Ontology Stream Reasoning with Truth Maintenance Systems**

- **Continuous ontology-based query answering**
  - C-SPARQL/SPARQL\textsubscript{stream}/CQEL Languages
    - done in the previous sessions
  - ETALIS and EP-SPARQL
  - Stream Reasoning with ASP
    - done in the previous section

- **Formal Semantics of Stream Reasoning**
  - LARS
  - STARQL
Ontology Stream Reasoning with TMS

- **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, Univ of Aberdeen 2011-2013

- **Reference**

- **Code:**
  - http://trowl.eu
  - Tutorial support, actively maintained
Ontology Stream Reasoning with TMS

- **Ontologies evolve over time!**
  - Adding and removing axioms over time.
  - **Ontology stream:** sequence of classical ontologies
    - \( O(0), O(1), ..., O(n) \)

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

\[
O(i+1) = O(i) \cup Ad(i) \setminus Er(i)
\]
Ontology Stream Reasoning with TMS

Answering queries on snapshots

- Re-compute every time is not efficient
- The DRed (Delete and Re-derive) approach [Volz et. al. 2005]
  - Maintaining the materialisation of the knowledge base
  - Over-delete impacted entailments
  - Re-derive impacted entailments

- Give me all talks interesting for David

New axioms over time
Justification: Given an ontology O and a reasoning result rs

A justification J(rs) is a **minimal subset of O** that imply rs

If the current justification J(rs) and Er(i) overlap:
- then rs should be removed as well

But...
- Computing one justification for OWL2-DL is costly
- Computing all justifications is NP-complete
Ontology Stream Reasoning with TMS
Truth Maintenance System

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

- All entailments are reachable from their justifications
  - Easy to identify impacted entailments
Ontology Stream Reasoning with TMS
Delete and re-derive

- **Erasing:**
  - Remove all nodes reachable from the erased axioms
  - Removing all corresponding edges

- **Adding:**
  - Adding added axioms as new nodes into the graph
  - Inferring new results
  - Establishing new edges
- TMS maintenance and computing justifications is expensive

- Optimised memory consumption
  - **Reduce** the number of maintained nodes and edges

- We entail an axiom \( C \sqsubseteq \exists r.D \) if it is **classification-relevant**, i.e. contributing to the reasoning results we are looking for:
  - E.g. there is some \( \exists r.A \sqsubseteq B \)
  - or, \( r \sqsubseteq s \) and \( s \) is classification-relevant
  - or, ...
- 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
Agenda

- Incremental Maintenance Materializations of Ontologies
  - IMaRS
    - done in the previous section
  - Sparkwave
  - DynamiTE: Parallel Materialization of Dynamic RDF Data
  - RDF Stream Reasoning with GPUs
  - Ontology Stream Reasoning with Truth Maintenance Systems

- Continuous ontology-based query answering
  - C-SPARQL/SPARQL\textsubscript{stream}/CQEL Languages
    - done in the previous sessions
  - **ETALIS and EP-SPARQL**
  - Stream Reasoning with ASP
    - done in the previous section

- Formal Semantics of Stream Reasoning
  - LARS
  - STARQL
ETALIS/EP-SPARQL

- **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:**
  - D.Anicic, S.Rudolph, P.Fodor, N.Stojanovic 2010-2012

- **References**

- **Code:**
  - https://code.google.com/p/etalis/
  - Tutorial support, actively maintained
Recursive CEP in ETALIS
- **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:

\[ P ::= \text{pr}(t_1, \ldots, t_n) \mid P \text{ WHERE } t \mid q \mid (P).q \mid P \text{ BIN } P \mid \text{NOT}(P) \cdot [P, P] \]

- \text{pr} – a predicate name with arity \( n \);
- \( t_i \) – denote terms;
- \( t \) – 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

\[ \text{pr}(t_1, \ldots, t_n) \]
## ETALIS

### Declarative Semantics

<table>
<thead>
<tr>
<th>pattern</th>
<th>$I_\mu(p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(t_1,\ldots, t_n)$</td>
<td>$I(p^<em>(t_1), \ldots, t^</em>(t_n))$</td>
</tr>
</tbody>
</table>
| $p$ WHERE $t$ | $I_\mu(p)$ if $p^*(t) = \text{true}$  
$\emptyset$ otherwise. |
| $q$ $(p).q$ | $I(p) \cap \{(q_1, q_2) \mid q_2 - q_1 = q\}$ |
| $p_1$ SEQUENTIAL $p_2$ | $\{(q_1, q_4) \mid \langle q_1, q_2 \rangle \in I_\mu(p_1) \text{ and } \langle q_3, q_4 \rangle \in I_\mu(p_2) \text{ for some } q_2, q_3 \in \mathbb{Q}^+ \text{ with } q_2 < q_3\}$ |
| $p_1$ AND $p_2$ | $\{(\min(q_1, q_3), \max(q_2, q_4)) \mid \langle q_1, q_2 \rangle \in I_\mu(p_1) \text{ and } \langle q_3, q_4 \rangle \in I_\mu(p_2) \text{ for some } q_2, q_3 \in \mathbb{Q}^+\}$ |
| $p_1$ PARALLEL $p_2$ | $\{(\min(q_1, q_3), \max(q_2, q_4)) \mid \langle q_1, q_2 \rangle \in I_\mu(p_1) \text{ and } \langle q_3, q_4 \rangle \in I_\mu(p_2) \text{ for some } q_2, q_3 \in \mathbb{Q}^+ \text{ with } \max(q_1, q_3) < \min(q_2, q_4)\}$ |
| $p_1$ OR $p_2$ | $I_\mu(p_1) \cup I_\mu(p_2)$ |
| $p_1$ EQUALS $p_2$ | $I_\mu(p_1) \cap I_\mu(p_2)$ |
| $p_1$ MEETS $p_2$ | $\{(q_1, q_3) \mid \langle q_1, q_2 \rangle \in I_\mu(p_1) \text{ and } \langle q_2, q_3 \rangle \in I_\mu(p_2) \text{ for some } q_2 \in \mathbb{Q}^+\}$ |
| $p_1$ DURING $p_2$ | $\{(q_3, q_4) \mid \langle q_1, q_2 \rangle \in I_\mu(p_1) \text{ and } \langle q_3, q_4 \rangle \in I_\mu(p_2) \text{ for some } q_2, q_3 \in \mathbb{Q}^+ \text{ with } q_3 < q_1 < q_2 < q_4\}$ |
| $p_1$ STARTS $p_2$ | $\{(q_1, q_3) \mid \langle q_1, q_2 \rangle \in I_\mu(p_1) \text{ and } \langle q_1, q_3 \rangle \in I_\mu(p_2) \text{ for some } q_2 \in \mathbb{Q}^+ \text{ with } q_2 < q_3\}$ |
| $p_1$ FINISHES $p_2$ | $\{(q_1, q_3) \mid \langle q_1, q_2 \rangle \in I_\mu(p_1) \text{ and } \langle q_1, q_3 \rangle \in I_\mu(p_2) \text{ for some } q_2 \in \mathbb{Q}^+ \text{ with } q_1 < q_2\}$ |

**Definition of extensional interpretation of event patterns.** We use $p(x)$ for patterns, $q(x)$ for rational numbers, $t(x)$ for terms and $pr$ for event predicates.
EP-SPARQL
Extended SPARQL interface to ETALIS

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

RDF stream – a set of \( \text{triple occurrences} \) \(<\langle s, p, o \rangle, t_\alpha, t_\omega \rangle \) where \(<s, p, o \rangle\) 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.,

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

Special functions
- getDuration(), getStartTime(), getEndTime()
**EP-SPARQL**

Extended SPARQL interface to ETALIS

- **Sequence operators and CEP world**

  - **SEQ**: joins $e_{t_i,t_f}$ and $e'_{t_i',t'_f}$ if $e'$ occurs after $e$
  - **EQUALS**: joins $e_{t_i,t_f}$ and $e'_{t_i',t'_f}$ if they occur simultaneously
  - **OPTIONALSEQ, OPTIONALEQUALS**: Optional join variants

**Diagram:**

- EP-SPARQL query → translator → Prolog engine → continuous results

---

http://streamreasoning.org/events/sr4ld2015
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.

```sparql
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 Maintenance Materializations of Ontologies
  - IMaRS
    - done in the previous section
  - Sparkwave
  - DynamiTE: Parallel Materialization of Dynamic RDF Data
  - RDF Stream Reasoning with GPUs
  - Ontology Stream Reasoning with Truth Maintenance Systems

- Continuous ontology-based query answering
  - C-SPARQL/SPARQL\textsubscript{stream}/CQEL Languages
    - done in the previous sessions
  - ETALIS and EP-SPARQL
  - Stream Reasoning with ASP
    - done in the previous section

- Formal Semantics of Stream Reasoning
  - LARS
  - STARQL
LARS

- **Addressed task:**
  - A formalization of continuous query answering over data streams

- **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
LARS Formulas

- Formula elements:
  - Window operators ⊞ (substream generation)
  - Boolean connectives: $\land$, $\lor$, $\rightarrow$, $\neg$
  - Temporal/modal operators: ◇, □, @_t

- Formulas are defined by the grammar:
  $$\alpha ::= a \mid \neg \alpha \mid \alpha \land \alpha \mid \alpha \lor \alpha \mid \alpha \rightarrow \alpha \mid \diamond \alpha \mid \Box \alpha \mid @_t \alpha \mid \box_i \alpha$$

Where:
- $\alpha$: $\alpha$ holds now
- Boolean connectives work as in first order logic
- ◇$\alpha$: $\alpha$ holds at some time instant in the past
- □$\alpha$: $\alpha$ 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 $\square_i^x \alpha$ is used to set the scope (substream) on which $\alpha$ applies.

$\square_i^x$ 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:

\[
\square_i^{60} \Diamond \square_i^{5} \Box \alpha
\]

(where $\square_i^{60}$ and $\square_i^{5}$ are two time-based sliding windows of 60 and 5 minutes)
LARS
Rules to generate intensional data (inference)

- 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: 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, ...)

\[ c \leftarrow \bigoplus (a \land \Diamond b) \]
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)
LARS
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)
Agenda

- Incremental Maintenance Materializations of Ontologies
  - IMaRS
    - done in the previous section
  - Sparkwave
  - DynamiTE: Parallel Materialization of Dynamic RDF Data
  - RDF Stream Reasoning with GPUs
  - Ontology Stream Reasoning with Truth Maintenance Systems

- Continuous ontology-based query answering
  - C-SPARQL/SPARQL$_{\text{stream}}$/CQEL Languages
    - done in the previous sessions
  - ETALIS and EP-SPARQL
  - Stream Reasoning with ASP
    - done in the previous section

- Formal Semantics of Stream Reasoning
  - LARS
  - STARQL
STARQL

- 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:
  - Hamburg University of Technology, 2013-ongoing

- (Some) Publications:
  - ÖL Özçep, R Möller, C Neuenstadt, “A Stream-Temporal Query Language for Ontology Based Data Access”. Description Logics, 2014
STARQL
A two-layer framework

- Streaming and Temporal ontology Access with a Reasoning-based Query Language
  - A framework to access and query heterogeneous sensor data through ontologies
  - STARQL follows the OBDA paradigm:
    - An ontology to give an holistic view over the static and streaming data
    - Query are composed using the ontology concepts
  - 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

http://streamreasoning.org/events/sr4ld2015
STARQL is a 2-layer framework

\[ \text{STARQL(OL,ECL)} \]

composed by:
- an Ontology Language (OL) to model the data and its schema
- an Embedded Constraint Language (ECL) to compose the queries

Examples:
- \[ \text{STARQL(DL-Lite,UCQ): Union of Conjunctive Queries over DL-Lite ontologies.} \]
  - FOL-rewritability property
- \[ \text{STARQL(SHI,GCQ): Grounded Conjunctive Queries over SHI ontologies} \]
  - 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 $\Theta_1(x,y)\langle timeExp_1\rangle,\ldots,\Theta_r(x,y)\langle timeExp_r\rangle$
FROM winExp_1,\ldots,S_m\; winExp_m,A^0_{st},\ldots,A^k_{st},T^0,\ldots,T^l
WHERE $\psi(x)$
SEQUENCE BY seqMeth
HAVING $\phi(x,y)$
```

STARQL introduces extensions to
- Define windows over the streams: $S_1\; winExp_1$
- Transform the streams in sequences of time-ordered Aboxes: $SEQUENCE\; BY\; seqMeth$
- Process those sequences: $HAVING\; \phi(x,y)$
- The output is a stream with the computed assertions
STARQL query semantics

Static ABoxes → WHERE clause → Bindings → Sequenced ontologies → HAVING clause → Output

Static TBoxes → WHERE clause → Bindings

Stream 1 → winExp₁ → + → joinStream

Stream m → winExpₘ
The query that detects the critical sensors in STARQL is the following:

```
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 EXISTS 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
```