Stream Reasoning
For Linked Data
M. Balduini, J-P Calbimonte, O. Corcho, D. Dell'Aglio, and E. Della Valle
http://streamreasoning.org/events/sr4ld2014

Other Stream Reasoning Approaches
Jean-Paul Calbimonte, Oscar Corcho, Daniele Dell'Aglio, Emanuele Della Valle
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

- To view a copy of this license, visit
  [http://creativecommons.org/licenses/by/3.0/](http://creativecommons.org/licenses/by/3.0/)
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_{stream}/CQEL Languages
  - done in the previous sessions
- ETALIS and EP-SPARQL
- Stream Reasoning with ASP
- Stream Reasoning with Probabilistic ASP

Formal Semantics of Stream Reasoning
- STARQL
- TU-Wien's Stream Reasoning Framework
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
  - Stream Reasoning with Probabilistic ASP

- Formal Semantics of Stream Reasoning
  - STARQL
  - TU-Wien's Stream Reasoning Framework
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**

- **References**
Basic principles: the RETE algorithm

- We will illustrate how Sparkwave works with the following basic SPARQL query:
  
  - SELECT ?x ?y WHERE
    
    ```sparql
    {?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 imagine the conjunctive query: $(?x \ a \ b) \wedge (?x \ c \ ?y) \wedge (?y \ m \ n)$
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.

```
RDF triple streams

\[
\begin{array}{c}
\text{RDF triple streams} \\
\mapsto \\
\text{RDF graph pattern instances}
\end{array}
\]

\[
\begin{array}{c}
\text{Schema} \\
\mapsto \\
\text{RDF graph pattern}
\end{array}
\]

\[
\begin{array}{c}
\text{Sparkwave network} \\
\mapsto \\
\text{Rete network}
\end{array}
\]
```
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 1: RDF/RDFS entailment rules

<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:type rdf:Property)</td>
</tr>
<tr>
<td>rdfs2</td>
<td>(p rdfs:domain c) (x p y)</td>
<td>(x rdf:type c)</td>
</tr>
<tr>
<td>rdfs3</td>
<td>(p rdfs:range c) (x p y)</td>
<td>(y rdf:type c)</td>
</tr>
<tr>
<td>rdfs4a</td>
<td>(x p y)</td>
<td>(x rdf:type rdfs:Resource)</td>
</tr>
<tr>
<td>rdfs4b</td>
<td>(x p y)</td>
<td>(y rdf:type rdfs:Resource)</td>
</tr>
<tr>
<td>rdfs5</td>
<td>(p rdfs:subPropertyOf q) (q rdfs:subPropertyOf r)</td>
<td>(p rdfs:subPropertyOf r)</td>
</tr>
<tr>
<td>rdfs6</td>
<td>(p rdf:type rdfs:Property)</td>
<td>(p rdfs:subPropertyOf p)</td>
</tr>
<tr>
<td>rdfs7</td>
<td>(p rdfs:subPropertyOf q) (x p y)</td>
<td>(x q y)</td>
</tr>
<tr>
<td>rdfs8</td>
<td>(c rdf:type rdfs:Class)</td>
<td>(c rdfs:subClassOf rdfs:Resource)</td>
</tr>
<tr>
<td>rdfs9</td>
<td>(c rdfs:subClassOf d) (x rdf:type c)</td>
<td>(x rdf:type d)</td>
</tr>
<tr>
<td>rdfs10</td>
<td>(c rdf:type rdfs:Class)</td>
<td>(c rdfs:subClassOf c)</td>
</tr>
<tr>
<td>rdfs11</td>
<td>(c rdfs:subClassOf d) (d rdfs:subClassOf e)</td>
<td>(c rdfs:subClassOf e)</td>
</tr>
<tr>
<td>rdfs12</td>
<td>(p rdf:type rdfs:ContainerMembershipProperty)</td>
<td>(p rdfs:subPropertyOf rdfs:member)</td>
</tr>
<tr>
<td>rdfs13</td>
<td>(x rdf:type rdfs:Datatype)</td>
<td>(x rdfs:subClassOf rdfs: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:inverseOf q)</td>
<td>(q owl:inverseOf p)</td>
</tr>
<tr>
<td>inv2</td>
<td>(p owl:inverseOf q) (x p y)</td>
<td>(y q x)</td>
</tr>
<tr>
<td>sym</td>
<td>(p rdf:type owl:SymmetricProperty) (x p y)</td>
<td>(y p x)</td>
</tr>
</tbody>
</table>

Including...
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: limitations

- Sparkwave operates over a fixed schema
  - The $\varepsilon$-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\textsubscript{stream}/CQEL Languages
    - done in the previous sessions
  - ETALIS and EP-SPARQL
  - Stream Reasoning with ASP
  - Stream Reasoning with Probabilistic ASP

- Formal Semantics of Stream Reasoning
  - STARQL
  - TU-Wien's Stream Reasoning Framework
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:**
  - Urbani, Margara, Jacobs et al. VUA Amsterdam.

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

Dynamite: Parallel Materialization

- **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:** $\rho_{DF}$
Dynamite Workflow

- Maintenance of an RDF database

- Key: Incremental Materialization

Maintain the KB when there are updates.

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 3 types of rules
Parallelize: 1 thread per rule

Divide in schema and generic triples
Materialization after removals

- Each triple with a count attribute:
  - number of possible rule instantiations that produced t as a direct consequence
- For more complex scenarios: iteratively

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>DRed</td>
<td>Counting</td>
<td></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
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

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
- Stream Reasoning with Probabilistic ASP

Formal Semantics of Stream Reasoning

- STARQL
- TU-Wien's Stream Reasoning Framework
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:**
  - Liu, Urbani, Qi. VUA Amsterdam, U Maryland, U Southeast China

- **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 \(t\):
    - \(\{\tau:[0, +\infty]| \tau \in KB\} \cup \{\tau:[t', t'+w]|(\tau, t') \in S\) 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
  - Stream Reasoning with Probabilistic ASP

- Formal Semantics of Stream Reasoning
  - STARQL
  - TU-Wien's Stream Reasoning Framework
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 OWL 2DL

- **Who:**
  - Yuan, Pan, Univ of Aberdeen

- **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 $O(i)$

$$O(i+1) = O(i) \cup \text{Ad}(i) \setminus \text{Er}(i)$$

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

Given axioms over time

Justifications for deletes

- 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

---

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

---

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
Stream reasoning for OWL2 EL

- 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, ...

---

Stream Reasoning for OWL2 DL

- Generate a TMS when doing approximation and reasoning
  - Nodes:
    - Asserted axioms;
    - Approximated axioms;
    - Entailed axioms;
  - Edges:
    - Created during approximation and reasoning

---

Discussion

- 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
  - \textbf{ETALIS and EP-SPARQL}
  - Stream Reasoning with ASP
  - Stream Reasoning with Probabilistic ASP

- Formal Semantics of Stream Reasoning
  - STARQL
  - TU-Wien's Stream Reasoning Framework
ETALIS and EP-SPARQL

- References
  - A Declarative Framework for Matching Iterative and Aggregative Patterns against Event Streams
    - Darko Anicic, Sebastian Rudolph, Paul Fodor, Nenad Stojanovic

Recursive CEP in ETALIS

ETALIS Features:
- Logic-based CEP
  - Stream (deductive) reasoning
- Iterative and aggregative patterns
- Implementation
  - http://code.google.com/p/etalis
- **Iterative 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 \( \text{pr}(t_1, \ldots, t_n) \).
ETALIS: Declarative Semantics

<table>
<thead>
<tr>
<th>pattern</th>
<th>$\mathcal{I}_\mu$ (pattern)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{pr}(t_1, \ldots, t_n)$</td>
<td>$\mathcal{I}(\text{pr}(\mu^<em>(t_1), \ldots, \mu^</em>(t_n)))$</td>
</tr>
<tr>
<td>$p \text{ WHERE } t$</td>
<td>$\mathcal{I}_\mu(p)$ if $\mu^*(t) = \text{true}$ $\emptyset$ otherwise.</td>
</tr>
<tr>
<td>$q$</td>
<td>${(q, q)}$ for all $q \in \mathbb{Q}^+$ $\emptyset$ otherwise.</td>
</tr>
<tr>
<td>$(p).q$</td>
<td>$\mathcal{I}_\mu(p) \cap {(q_1, q_2) \mid q_2 - q_1 = q}$</td>
</tr>
<tr>
<td>$p_1 \text{ SEQ } p_2$</td>
<td>${(q_1, q_4) \mid (q_1, q_2) \in \mathcal{I}<em>\mu(p_1) \text{ and } (q_3, q_4) \in \mathcal{I}</em>\mu(p_2) \text{ for some } q_2, q_3 \in \mathbb{Q}^+ \text{ with } q_2 &lt; q_3}$</td>
</tr>
<tr>
<td>$p_1 \text{ AND } p_2$</td>
<td>${(\min(q_1, q_3), \max(q_2, q_4)) \mid (q_1, q_2) \in \mathcal{I}<em>\mu(p_1) \text{ and } (q_3, q_4) \in \mathcal{I}</em>\mu(p_2) \text{ for some } q_2, q_3 \in \mathbb{Q}^+}$</td>
</tr>
<tr>
<td>$p_1 \text{ PAR } p_2$</td>
<td>${(\min(q_1, q_3), \max(q_2, q_4)) \mid (q_1, q_2) \in \mathcal{I}<em>\mu(p_1) \text{ and } (q_3, q_4) \in \mathcal{I}</em>\mu(p_2) \text{ for some } q_2, q_3 \in \mathbb{Q}^+ \text{ with } \max(q_1, q_3) &lt; \min(q_2, q_4)}$</td>
</tr>
<tr>
<td>$p_1 \text{ OR } p_2$</td>
<td>$\mathcal{I}<em>\mu(p_1) \cup \mathcal{I}</em>\mu(p_2)$</td>
</tr>
<tr>
<td>$p_1 \text{ EQUALS } p_2$</td>
<td>$\mathcal{I}<em>\mu(p_1) \cap \mathcal{I}</em>\mu(p_2)$</td>
</tr>
<tr>
<td>$p_1 \text{ MEETS } p_2$</td>
<td>${(q_1, q_3) \mid (q_1, q_2) \in \mathcal{I}<em>\mu(p_1) \text{ and } (q_2, q_3) \in \mathcal{I}</em>\mu(p_2) \text{ for some } q_2 \in \mathbb{Q}^+}$</td>
</tr>
<tr>
<td>$p_1 \text{ DURING } p_2$</td>
<td>${(q_3, q_4) \mid (q_1, q_2) \in \mathcal{I}<em>\mu(p_1) \text{ and } (q_3, q_4) \in \mathcal{I}</em>\mu(p_2) \text{ for some } q_2, q_3 \in \mathbb{Q}^+ \text{ with } q_3 &lt; q_1 &lt; q_2 &lt; q_4}$</td>
</tr>
<tr>
<td>$p_1 \text{ STARTS } p_2$</td>
<td>${(q_1, q_3) \mid (q_1, q_2) \in \mathcal{I}<em>\mu(p_1) \text{ and } (q_1, q_3) \in \mathcal{I}</em>\mu(p_2) \text{ for some } q_2 \in \mathbb{Q}^+ \text{ with } q_2 &lt; q_3}$</td>
</tr>
<tr>
<td>$p_1 \text{ FINISHES } p_2$</td>
<td>${(q_1, q_3) \mid (q_2, q_3) \in \mathcal{I}<em>\mu(p_1) \text{ and } (q_1, q_3) \in \mathcal{I}</em>\mu(p_2) \text{ for some } q_2 \in \mathbb{Q}^+ \text{ with } q_1 &lt; q_2}$</td>
</tr>
<tr>
<td>$\text{NOT}(p_1).[p_2, p_3]$</td>
<td>$\mathcal{I}<em>\mu(p_2 \text{ SEQ } p_3) \setminus \mathcal{I}</em>\mu(p_2 \text{ SEQ } p_1 \text{ SEQ } p_3)$</td>
</tr>
</tbody>
</table>

Definition of extensional interpretation of event patterns. We use $p(x)$ for patterns, $q(x)$ for rational numbers, $t(x)$ for terms and $\text{pr}$ for event predicates.
EP-SPARQL (I)

- Basics
  - SPARQL extension (as with other previously seen languages)
  - Interval-based: 2 timestamps
    
    RDF stream – a set of triple occurrences $\langle \langle s, p, o \rangle, t_\alpha, t_\omega \rangle$ where $\langle s, p, o \rangle$ 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.,
      
      AND – joins $\langle \mu, t_\alpha, t_\omega \rangle$ and $\langle \mu', t'_\alpha, t'_\omega \rangle$. 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

Diagram:

- $e_1$ at 1
- $e_2$ at 3
- $e_3$ at 6
- $e_4$ at 9

**Sequence**

**Simultaneous**

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.

```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)
```
EP-SPARQL sample translation (SEQ)

```
SELECT ?company WHERE
{
  ?comp hasStockPrice ?pr1
}
SEQ { ?comp hasStockPrice ?pr2 }
SEQ { ?comp hasStockPrice ?pr3 }

⟨⟨s, p, o⟩, tᵢ, tⱼ⟩ represented as triple(s, p, o, Tᵢ, Tⱼ), and τ represents s, p, o.

triple(τᵢ, T₁, T₄) ← triple(τ₁, T₁, T₂) SEQ triple(τ₂, T₃, T₄).
triiple(τ, T₁, T₆) ← triple(τᵢ, T₁, T₄) SEQ triple(τ₃, T₅, T₆).
```

Rule transformation – Incremental computation (Prolog syntax)

```
triiple(τ₁, T₁, T₂) :-
  assert (goal(triiple(τ₂, _, _), triiple(τ₁, T₁, T₂), triiple(τᵢ, _, _))).

triiple(τ₂, T₃, T₄) :-
  goal(triiple(τ₂, _, _), triiple(τ₁, T₁, T₂), triiple(τ, _, _)),
  T₂ < T₃,
  retract (goal(triiple(τ₂, _, _), triiple(τ₁, T₁, T₂), triiple(τᵢ, _, _))),
  triiple(τᵢ, T₁, T₄).
```
Examples

A sum over an unbound event stream until a threshold value is met:

\[
\text{income}(Price) \leftarrow \text{sell}(Item, Price).
\]
\[
\text{income}(P1 + P2) \leftarrow \text{income}(P1) \text{ SEQ sell}(Item, P2).
\]
\[
\text{bigincome} \leftarrow \text{income}(Price) \text{ WHERE Price} > 100000.
\]

The k-fold sequential execution of an event a:

\[
\text{iteration}(a, 1) \leftarrow a.
\]
\[
\text{iteration}(a, k + 1) \leftarrow a \text{ SEQ iteration}(a, k).
\]

A sliding length-based window, e.g., n=5:

\[
\text{iteration}(a, 1) \leftarrow a.
\]
\[
\text{iteration}(a, k + 1) \leftarrow \text{NOT}(a).[a, \text{iteration}(a, k)].
\]
\[
e \leftarrow \text{iteration}(a, n).
\]
1. Complex pattern (not event-driven rule)

\[ a \text{ SEQ } b \text{ SEQ } c \rightarrow \text{ce1} \]

2. Decoupling

\[(a \text{ SEQ } b) \text{ SEQ } c \rightarrow \text{ce1}\]

3. Binarization

\[ a \text{ SEQ } b \rightarrow \text{ie} \]
\[ \text{ie SEQ } c \rightarrow \text{ce1} \]

4. Event-driven backward chaining rules

```
Algorithm 2 Conjunction.

Input: event binary goal \(\text{ie} \leftarrow a \text{ AND } b\).
Output: event-driven backward chaining rules for AND operator.

Each event binary goal \(\text{ie} \leftarrow a \text{ AND } b\) is converted into:

\[
a(T_1, T_2) : \neg \text{for_each}(a, 1, [T_1, T_2]).
\]
\[
a(1, T_3, T_4) : \neg \text{goal}(a(\_, \_), b(T_1, T_2), \text{ie}(\_, \_)),
\]
\[
\text{retract}(\text{goal}(a(\_, \_), b(T_1, T_2), \text{ie}(\_, \_))),
\]
\[
T_5 = \min\{T_1, T_3\}, T_6 = \max\{T_2, T_4\}, \text{ie}(T_5, T_6).
\]
\[
a(2, T_3, T_4) : \neg(\text{goal}(a(\_, \_), b(T_1, T_2), \text{ie}(\_, \_))),
\]
\[
\text{assert}(\text{goal}(\text{ie}(\_, \_), a(T_3, T_4), \text{ie}(\_, \_))).
\]
\[
b(T_1, T_2) : \neg \text{for_each}(b, 1, [T_1, T_2]).
\]
\[
b(1, T_3, T_4) : \neg \text{goal}(\text{ie}(\_, \_), a(T_1, T_2), \text{ie}(\_, \_)),
\]
\[
\text{retract}(\text{goal}(\text{ie}(\_, \_), a(T_1, T_2), \text{ie}(\_, \_))),
\]
\[
T_5 = \min\{T_1, T_3\}, T_6 = \max\{T_2, T_4\}, IE(T_5, T_6).
\]
\[
b(2, T_3, T_4) : \neg(\text{goal}(\text{ie}(\_, \_), a(T_1, T_2), \text{ie}(\_, \_))),
\]
\[
\text{assert}(\text{goal}(a(\_, \_), b(T_3, T_4), \text{ie}(\_, \_))).
\]
```
For any aggregate function, calculated over a window, we implement the following three rules:

\[
\begin{align*}
\text{iteration}(\text{StartCntr} = 0, \text{StartVal}) & \leftarrow \text{start\_event}(\text{StartVal}). \\
\text{iteration}(\text{OldCntr} + 1, \text{NewVal}) & \leftarrow \\
& \text{iteration}(\text{OldCntr}, \text{OldVal}) \text{ SEQ } a(\text{AggArg}) \text{ WHERE } \{ \text{assert}(\text{AggArg}), \\
& \text{window}(\text{WndwSize}, \text{OldCntr}, \text{OldVal}, \text{AggArg}, \text{NewVal}) \}. \\
\text{window}(\text{WndwSize}, \text{OldCntr}, \text{OldVal}, \text{AggArg}, \text{NewVal}) & : = \\
& \text{OldCntr} + 1 \geq \text{WindowSize} \rightarrow \\
& \text{retract}(\text{LastItem}), \\
& \text{spec\_aggregate}(\text{OldValue}, \text{AggArg}, \text{NewValue}); \\
& \text{spec\_aggregate}(\text{OldValue}, \text{AggArg}, \text{NewValue}).
\end{align*}
\]
SUM aggregate function:

\[
\begin{align*}
\text{sum}(\text{StartCntr} = 0, \text{StartVal}) & \leftarrow \text{start\_event}(\text{StartVal}). \\
\text{sum}(\text{OldCntr} + 1, \text{NewSum}) & \leftarrow \\
& \quad \text{sum}(\text{OldCntr} + 1, \text{OldSum}) \ \text{SEQ} \ a(\text{AggArg}) \\
& \quad \text{WHERE} \ \{ \text{assert}(\text{AggArg}), \\
& \quad \quad \text{window}(\text{WndwSize}, \text{OldCntr}, \\
& \quad \quad \quad \text{OldSum} + \text{AggArg}, \text{AggArg}, \text{NewSum}) \}. \\
\text{window}(\text{WndwSize}, \text{OldCntr}, \text{CurrSum}, \text{NewSum}) : - \\
& \quad \text{OldCntr} + 1 \geq \text{WindowSize} \rightarrow \\
& \quad \text{retract}(\text{LastItem}), \\
& \quad \text{NewSum} = \text{CurrSum} - \text{LastItem}; \\
& \quad \text{NewSum} = \text{CurrSum} - \text{LastItem}.
\end{align*}
\]
MAX aggregate function:

\[
\begin{align*}
\text{max}(\text{StartCntr} = 0, \text{StartVal}) & \leftarrow \text{start\_event}(\text{StartVal}). \\
\text{max}(\text{OldCntr} + 1, \text{NewMax}) & \leftarrow \\
& \quad \text{max}(\text{OldCntr} + 1, \text{OldMax}) \text{ SEQ a(\text{AggArg})} \\
& \quad \text{WHERE \{assert(\text{AggArg}),} \\
& \quad \quad \text{window(WndwSize, OldCntr, NewMax)\}}. \\
\text{window(WndwSize, OldCntr, NewMax)} : & = \\
& \quad \text{OldCntr} + 1 \geq \text{WindowSize} \rightarrow \\
& \quad \text{retract(LastItem), get(NewMax);} \\
& \quad \text{get(NewMax).}
\end{align*}
\]
COUNT aggregate function:

\[
\text{iteration}(\text{StartCntr} = 0, \text{StartVal}) \leftarrow \text{start_event}(\text{StartVal}).
\]
\[
\text{iteration}(\text{NewCntr}) \leftarrow
\text{iteration}(\text{OldCntr}) \text{ SEQ } a(\text{AggArg})
\text{ WHERE } \{ \text{NewCntr} = \text{getCount}([T_2, T_1]), \text{window}(3\text{min}) \}.
\]

- Data structures: red-black trees, stack, difference lists
- Time-based windows require a time-based garbage collection (GC);
- We have implemented two techniques for GC:
  - pushed constraints
  - general and pattern-based GC
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_{stream}/CQEL Languages
    - done in the previous sessions
  - ETALIS and EP-SPARQL
  - **Stream Reasoning with ASP**
  - Stream Reasoning with Probabilistic ASP

- Formal Semantics of Stream Reasoning
  - STARQL
  - TU-Wien's Stream Reasoning Framework
Goal:
- Extend the Stream Reasoning approach to exploratory search dynamic problems

Key contributions:
- Formal semantics of Stream Reasoning with Answer Set Programming (ASP)
- ASP solver for exploratory search dynamic problems encoded as incremental ASP rules

Who:
- M. Gebser, T. Grote, R. Kaminski, O. Sabuncu, T. Schaub
  Potsdam University, Germany
- P. Obermeier, Insight Galway, Ireland

Code:
- Clingo solver on http://potassco.sourceforge.net/
- Tutorial support, actively maintained
Stream Reasoning with ASP 2/2

- References
  - Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, Sven Thiele: Engineering an Incremental ASP Solver. ICLP 2008: 190-205
In *Answer Set Programming (ASP)*, the problem is encoded in terms of a logic theory - typically using logic programming rules.

- Answer-Set Solver computes dedicated models of the theory.
- They are called *answer sets*.
- Correspond one-to-one to the (multiple) solutions of the problem.
Typically, answer-set programs are *uniform problem encodings*.

The program itself encodes the general problem, whereas individual problem instances represented by facts can be joined with the program on demand.

```
Problem Instance                      General Problem
F₁ ←                                H₁ ← B₁
...                                  ...
Fₗ ←                                Hₙ ← Bₙ

Set of Facts          Answer-Set Program
∪                       

Solver

Answer Set 1          Answer Set m
←                     ←
Solution 1            Solution m
```
Problem Encoding, e.g., graph colouring 1/2

Uniform problem encoding:

\[ P_1 = \{ \text{another\_col}(V, C) \leftarrow \text{vertex}(V), \text{col}(C), \text{col}(D), \text{col\_of}(V, D), C \neq D, \}
\]

\[ \text{col\_of}(V, C) \leftarrow \text{vertex}(V), \text{col}(C), \text{not another\_col}(V, C), \]

\[ \leftarrow \text{vertex}(U), \text{vertex}(V), \text{edge}(U, V), \text{col\_of}(U, C), \text{col\_of}(V, C) \}. \]

Input instance as set of facts:

\[ F_1 = \{ \text{col(red)}, \text{col(green)}, \text{col(blue)}, \text{vertex}(a), \text{vertex}(b), \text{vertex}(c), \text{vertex}(d), \text{vertex}(e), \text{edge}(a, b), \text{edge}(a, c), \text{edge}(a, d), \text{edge}(b, e), \text{edge}(c, d), \text{edge}(d, e) \}. \]
The answer sets of $P_1 \cup F_1$ correspond to the solutions of the problem instance.

$$\{\text{col}_\text{of}(a, \text{red}), \text{col}_\text{of}(b, \text{blue}), \text{col}_\text{of}(c, \text{blue}), \text{col}_\text{of}(d, \text{green}), \text{col}_\text{of}(e, \text{red}), \ldots \}$$
Incremental Answer Set Programming

- **Problem**
  - many real-world applications, like planning or model checking, comprise parameters reflecting solution sizes

- **ASP Solution**
  - deal with them by considering in turn one problem instance after another
    - gradually increasing the bound on the solution size.
  - highly inefficient

- **Goal**
  - Avoiding redundancy by gradually processing the extensions to a problem rather than repeatedly re-processing the entire extended problem.

- **Proposal**
  - *Incremental ASP*
Incremental Answer Set Programming

- Parametrized problem description
  - Base: B Static knowledge
  - Cumulative: P[k] Knowledge cumulating increasing the size k
  - Volatile: Q[k] Query whose answer changes for each k

- E.g.

  \[ a \text{ causes } p \]
  \[ \text{exogenous } a \]
  \[ \text{inertial } p \]

  \[ \neg p \text{ holds at 0} \]
  \[ p \text{ holds at } n \]
  \[ \neg a \text{ occurs at } n \]

  \[ B = \{ \]
  \[ p(0) \leftarrow \neg p(0) \]
  \[ \neg p(0) \leftarrow \neg p(0) \]
  \[ \leftarrow p(0), \neg p(0) \}\]

  \[ P[k] = \{ \]
  \[ a(k) \leftarrow \neg a(k) \]
  \[ \neg a(k) \leftarrow \neg a(k) \]
  \[ p(k) \leftarrow a(k) \]
  \[ p(k) \leftarrow p(k-1), \neg p(k) \]
  \[ \neg p(k) \leftarrow \neg p(k-1), \neg p(k) \]
  \[ \leftarrow p(k), \neg p(k) \]
  \[ \leftarrow a(k), \neg a(k) \}\]

  \[ Q[k] = \{ \]
  \[ \leftarrow \neg p(0) \]
  \[ \leftarrow \neg p(k) \]
  \[ \leftarrow \neg \neg a(k) \}\]

- No answer set at 1
- Answer set at 2 \{-p(0), a(1), p(1), \neg a(2), p(2)\}
Stream Reasoning with ASP

- **Problem**
  - time-decaying data poses a major challenge to ASP given that fixed encodings must tolerate emerging as well as expiring data.

- **Solution**
  - **Time-Decaying Logic Programs** extends incremental ASP
    - **Base**: B Static knowledge
    - **Cumulative**: P[t] Knowledge cumulating *until time t*
    - **External**: incorporate external information
    - **Volatile**: Q^n[t] query whose answer changes for each time t considering a time span of *n* steps
Stream Reasoning with ASP - Access Control

- Problem
  - users attempting to logging in a website.
  - access attempts can be denied or granted depending on the supplied password
  - a user account is (temporarily) closed in case of three access denials in a row

- Data

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>[1]</td>
<td>[1]</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>bob</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Legenda
- \([n]\) granted
- \(n\) denied
- \(n\) account (temporarily) closed

Note: timestamps deviate from the step \(i\) by at most 2
#const window=3. #const offset=2. #const denial=3. #iinit 1-offset.

#base.
user(bob; alice; claudie). % some users
signal(denied; granted). % some signals
{ account(U,closed) : user(U) }.
account(U,open) :- user(U), not account(U,closed).

#cumulative t.
#external access(U,S,t+offset) : user(U) : signal(S).
denied(U,1, t) :- access(U,denied,t+offset).
denied(U,N+1, t) :- access(U,denied,t+offset),
               denied(U,N,t-1), N < denial.
denied(U,denial,t) :- denied(U,denial,t-1).
              :- denied(U,denial,t), not account(U,closed).

#volatile t.
:- account(U,closed), not denied(U,denial,t).
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
  - **Stream Reasoning with Probabilistic ASP**

- **Formal Semantics of Stream Reasoning**
  - STARQL
  - TU-Wien's Stream Reasoning Framework
Stream Reasoning with Probabilistic ASP

- **Goal:**
  - Handle the "noisy nature" of data streams

- **Key contributions:**
  - Annotation of first-order formulas as well as ASP rules and facts with probabilities
  - Learning of such weights from examples (parameter estimation)
  - Combination of various contemporary AI techniques

- **Who:**
  - M. Nickles, A. Mileo, Insight Galway, Ireland

- **References:**
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
  - Stream Reasoning with Probabilistic ASP

- Formal Semantics of Stream Reasoning
  - \textbf{STARQL}
  - TU-Wien's Stream Reasoning Framework
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
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
A two-layer framework

- 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:

\[
\text{CONSTRUCT } \Theta_1(x,y)\langle \text{timeExp}_1 \rangle, \ldots, \Theta_r(x,y)\langle \text{timeExp}_r \rangle \\
\text{FROM } \text{winExp}_1, \ldots, \text{winExp}_m, A^0_{st}, \ldots, A^k_{st}, T^0, \ldots, T^l \\
\text{WHERE } \psi(x) \\
\text{SEQUENCE BY seqMeth} \\
\text{HAVING } \varphi(x,y)
\]

STARQL introduces extensions to

- Define windows over the streams: $S^1 \text{winExp}_1$
- Transform the streams in sequences of time-ordered Aboxes: $\text{SEQUENCE BY seqMeth}$
- Process those sequences: $\text{HAVING } \varphi(x,y)$
- The output is a stream with the computed assertions
STARQL – query semantics

Static ABoxes ➔ WHERE clause

Static TBoxes ➔ Bindings

Stream 1 ➔ winExp₁

Stream m ➔ winExpₘ

winExp₁ ➔ + ➔ joinStream

Bindings ➔ Sequenced ontologies ➔ HAVING clause ➔ CONSTRUCT clause ➔ Output

SEQ clause

CONSTRUCT clause

HAVING clause

Bindings

Sequenced ontologies

CONSTRUCT clause

Output
The query that detects the critical sensors in STARQL is the following:

CREATE STREAM Sout AS
CREATE PULSE AS START = 0s, FREQUENCY = 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
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
  - Stream Reasoning with Probabilistic ASP

- Formal Semantics of Stream Reasoning
  - STARQL
  - TU-Wien's Stream Reasoning Framework
Stream Reasoning Framework

- **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:**
The introduction of two operators to compose logical formulas:
- extract: @
- window: ⊞

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:
- \( a \): \( a \) holds now
- \( \Diamond \alpha \): \( \alpha \) 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 \( t \)
Window

- By default, a formula $\alpha$ refers to the whole stream content.
- The window $\square_i \alpha$ is used to set the scope on which $\alpha$ applies.
- $\square_i$ is a reference to a window function (identified by $i$) that, given a time instant and a stream, generates a substream.
  - 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:
  \[
  \square_60 \Diamond \square_5 \Box \alpha
  \]
  (where $\square_60$ and $\square_5$ are two time-based sliding windows of 60 and 5 minutes.)
Stream Reasoning Framework

- The framework is a novel work, but some results has already been achieved:
  - It tackles the problem of formalising the stream reasoning concepts
  - It proposes a formal semantics
  - It captures part of CQL (but not yet aggregates and orders)

- Next steps:
  - Capture a larger set of CQL
  - Capture non-window systems (e.g. E-TALIS)
  - Application of the framework to capture and compare existing Stream Reasoning approaches
Stream Reasoning For Linked Data
M. Balduini, J-P Calbimonte, O. Corcho, D. Dell'Aglio, and E. Della Valle
http://streamreasoning.org/events/sr4ld2014

Other Stream Reasoning Approaches
Jean-Paul Calbimonte, Oscar Corcho, Daniele Dell'Aglio, Emanuele Della Valle