Stream Reasoning For Linked Data
M. Balduini, J-P Calbimonte, O. Corcho, D. Dell'Aglio, E. Della Valle, and J.Z. Pan
http://streamreasoning.org/sr4ld2013

Other Stream Reasoning approaches
Oscar Corcho, Emanuele Della Valle
ocorcho@fi.upm.es, emanuele.dellavalle@polimi.it
This work is licensed under the Creative Commons Attribution 3.0 Unported License.

You 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
- “[source http://streamreasoning.org/sr4ld2013]” at the end of each reused slide
- a credits slide stating
  - These slides are partially based on “Streaming Reasoning for Linked Data 2013” by M. Balduini, J-P Calbimonte, O. Corcho, D. Dell'Aglio, E. Della Valle, and J.Z. Pan http://streamreasoning.org/sr4ld2013

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

- ETALIS and EP-SPARQL
  - A Declarative Framework for Matching Iterative and Aggregative Patterns against Event Streams
    - Darko Anicic, Sebastian Rudolph, Paul Fodor, Nenad Stojanovic
ETALIS Features:

- Logic-based CEP
- Stream (deductive) reasoning
- Iterative and aggregative patterns
- Implementation
  - [http://code.google.com/p/etalis](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 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: Interval-based Semantics

- \( P_1 \)
- \( P_2 \)
- \( P_3 \)
- \((P_1).3\)
- \( P_1 \text{ SEQ } P_3 \)
- \( P_2 \text{ AND } P_3 \)
- \( P_1 \text{ PAR } P_2 \)
- \( P_2 \text{ OR } P_3 \)
- \( P_1 \text{ DURING } (0 \text{ SEQ } 6) \)
- \( P_3 \text{ STARTS } P_1 \)
- \( P_1 \text{ EQUALS } P_3 \)
- \( \text{NOT}(P_3).[P_1, P_1] \)
- \( P_3 \text{ FINISHES } P_2 \)
- \( P_2 \text{ MEETS } P_3 \)
ETALIS: Declarative Semantics

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

- **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

The slide also includes a diagram illustrating sequence and simultaneous events with labels $e_1, e_2, e_3, e_4$.
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)

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

⟨⟨s, p, o⟩, t_i, t_j⟩ represented as triple(s, p, o, T_i, T_j), and τ represents s, p, o.

triple(τ_i, T_1, T_4) ← triple(τ_1, T_1, T_2) SEQ triple(τ_2, T_3, T_4).
triple(τ, T_1, T_6) ← triple(τ_i, T_1, T_4) SEQ triple(τ_3, T_5, T_6).
```

**Rule transformation – Incremental computation (Prolog syntax)**

```
triple(τ_1, T_1, T_2) :-
   assert(goal(triple(τ_2, _, _), triple(τ_1, T_1, T_2), triple(τ_i, _, _))).

triple(τ_2, T_3, T_4) :-
   goal(triple(τ_2, _, _), triple(τ_1, T_1, T_2), triple(τ, _, _)),
   T_2 < T_3,
   retract(goal(triple(τ_2, _, _), triple(τ_1, T_1, T_2), triple(τ_i, _, _))),
   triple(τ_i, T_1, T_4).
```
Stream Reasoning
For Linked Data
M. Balduini, J-P Calbimonte, O. Corcho, D. Dell'Aglio, E. Della Valle, and J.Z. Pan
http://streamreasoning.org/sr4ld2013

Other Stream Reasoning approaches
Oscar Corcho, Emanuele Della Valle
ocorcho@fi.upm.es, emanuele.dellavalle@polimi.it