Agenda

Introduction, Motivation

- ETALIS Language for Events
  - Syntax;
  - Semantics;
  - Experimental Results;
- Conclusion.
Use background (contextual) knowledge to explore **semantic relations** between events, and detect otherwise undetectable **complex situation**.

\[ \psi = ? \]
\[ \pi \leftarrow \varphi \lor \neg \psi. \]
\[ \varphi = \text{true}. \]

**CEP with on-the-fly knowledge evaluation and stream reasoning:**

- Complex situation based on **explicit** data (events) and **implicit/explicit** knowledge
- Classification and filtering
- Context evaluation
- Intelligent recommendation
- Predictive analysis
Today’s CEP systems are focused mostly on **throughput** and **timeliness**;
- Time critical actions/decisions are supposed to be taken upon detection of complex events;
- These actions additionally require evaluation of background knowledge;
- Knowledge captures the domain of interest or context related to actions/decisions;
- The task of reasoning over streaming data (events) and the background knowledge constitutes a challenge known as **Stream Reasoning**.

Current CEP systems provide **on the-fly analysis** of data streams, but mainly fall short when it comes to combining streams with evolving **knowledge** and performing **reasoning** tasks.
Motivation

Non-blocking Event Revision – Transactional Events

• Events in today’s CEP systems are assumed to be **immutable** and therefore **always correct**;
• In some situations however revisions are required:
  • an event was reported by mistake, but did not happen in reality;
  • an event was triggered and later revoked due to a transaction failure.
• As recognised in [Ryvkina et al. ICDE’06], event stream sources may issue **revision tuples** that amend previously issued events.

Current CEP systems provide **on** the-fly analysis of data streams, but typically don’t take these **revision** tuples into account and produce **correct** revision outputs.
DSMS Approaches for retractions in CEP:

- D. Carney et al. Monitoring streams: a new class of data management applications. In VLDB’02
  - based on archives of recent data and replying
  - whole recent history is kept archived
  - based on blocking, buffering and synchronisation point

Stream Reasoning approaches:

Agenda

- Introduction, Motivation

ETALIS: Retractable CEP and Stream Reasoning
  - Syntax;
  - Semantics;
  - Experimental Results;

- Conclusion.
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: Interval-based Semantics
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Interpretation</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 \ 	ext{WHERE} \ t)</td>
<td>(I_\mu(p) \ 	ext{if } \mu^*(t) = \text{true} )   (\emptyset) otherwise.</td>
</tr>
<tr>
<td>(q)</td>
<td>({\langle q, q \rangle} \ 	ext{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 \ 	ext{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 \ 	ext{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 \ 	ext{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 \ 	ext{OR} \ p_2)</td>
<td>(I_\mu(p_1) \cup I_\mu(p_2))</td>
</tr>
<tr>
<td>(p_1 \ 	ext{EQUALS} \ p_2)</td>
<td>(I_\mu(p_1) \cap I_\mu(p_2))</td>
</tr>
<tr>
<td>(p_1 \ 	ext{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 \ 	ext{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 \ 	ext{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 \ 	ext{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>(\neg(p_1).[p_2, p_3])</td>
<td>(I_\mu(p_2 \ 	ext{SEQ} \ p_3) \setminus I_\mu(p_2 \ 	ext{SEQ} \ p_1 \ 	ext{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.
ETALIS: Operational Semantics (SEQ)

1. Complex pattern (not event-driven rule)
2. Decoupling
3. Binarization
4. Event-driven backward chaining rules

**Algorithm 1** Sequence.

**Input:** event binary goal \( \text{ie} \leftarrow a \text{ SEQ } b \).

**Output:** event-driven backward chaining rules for SEQ operator.

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

\[
\begin{align*}
  a(T_1, T_2) & : \neg \text{for}_\text{each}(a, 1, [T_1, T_2]). \\
  a(1, T_1, T_2) & : \neg \text{assert}(\text{goal}(b(_, _), a(T_1, T_2), \text{ie}(_, _))). \\
  b(T_3, T_4) & : \neg \text{for}_\text{each}(b, 1, [T_3, T_4]). \\
  b(1, T_3, T_4) & : \neg \text{goal}(b(T_3, T_4), a(T_1, T_2), \text{ie}), T_2 < T_3, \\
                     & \text{retract}(\text{goal}(b(T_3, T_4), a(T_1, T_2), \text{ie}(_, _))), \text{ie}(T_1, T_4).
\end{align*}
\]
Algorithm 5 Sequence with retraction.

**Input:** event binary goal $ie_1 \leftarrow a \text{ SEQ } b$.

**Output:** event-driven backward chaining rules for SEQ operator including retraction.

Each event binary goal $ie_1 \leftarrow a \text{ SEQ } b$ is converted into:

\[
\begin{align*}
    a(ID, [T_1, T_2]) : & \quad \text{for each}(a, 1, ID, [T_1, T_2]). \\
    a(1, ID, [T_1, T_2]) : & \quad \text{assert}(\text{goal}(b(_, [_, _]), a(ID, [T_1, T_2]),
        ie_1(_, [_, _]))). \\
    \text{rev}_a(ID, [T_3, T_4]) : & \quad \text{for each}(\text{rev}_a, 1, ID, [T_3, T_4]). \\
    \text{rev}_a(1, ID, [T_3, T_4]) : & \quad \text{goal}(b(_, [_, _]), a(ID, [T_1, T_2]),
        ie_1(_, [_, _])), \text{retract}(\text{goal}(b(_, [_, _]),
        a(ID, [T_1, T_2]))). \\
    \text{rev}_a(2, ID, [T_3, T_4]) : & \quad (ie_1(ID, [T_1, T_2]),
        \text{retract}(ie_1(ID, [T_1, T_2])), \text{rev}_ie_1(ID, [T_1, T_2])); \text{true}. \\
    b(ID, [T_3, T_4]) : & \quad \text{for each}(b, 1, ID, [T_3, T_4]). \\
    b(1, ID, [T_3, T_4]) : & \quad \text{goal}(b(_, [_, _]), a(ID, [T_1, T_2]),
        ie_1(_, [_, _])), T_2 < T_3, ie_1(ID, [T_1, T_4]). \\
    \text{rev}_b(ID, [T_5, T_6]) : & \quad \text{for each}(\text{rev}_b, 1, ID, [T_5, T_6]). \\
    \text{rev}_b(1, ID, [T_5, T_6]) : & \quad (ie_1(ID, [T_1, T_4]),
        \text{retract}(ie_1(ID, [T_1, T_4])), \text{rev}_ie_1(ID, [T_1, T_4])); \text{true}. \\
    ie_1(ID, [T_1, T_4]) : & \quad \text{for each}(ie_1, 1, ID, [T_1, T_4]). \\
    ie_1(1, ID, [T_1, T_4]) : & \quad \text{assert}(ie_1(ID, [T_1, T_4])). \\
    \text{ie}_1(1, ID, [T_1, T_4]) : & \quad \text{assert}(\text{ie}_1(ID, [T_1, T_4])).
\end{align*}
\]
Tests I: CEP with Stream Reasoning

trendIncrease() ← (stockIcr(CompanyA) SEQ stockIcr(CompanyB)).10
AND inSupChain(CompanyA, CompanyB).

inSupChain(X, Y) ← linked(X, Y).
inSupChain(X, Z) ← linked(X, Y) AND inSupChain(Y, Z).

linked(CompanyA, CompanyB)
...
linked(CompanyY, CompanyZ)

Intel Core Quad CPU Q9400
2.66GHz, 8GB of RAM, Vista x64;
ETALIS on SWI Prolog 5.6.64 and
YAP Prolog 5.1.3 vs. Esper 3.3.0
Tests II: Throughput Comparison

\[ e(ID) \leftarrow a(ID) \text{ BIN } b(ID). \]

\[ e(ID) \leftarrow \text{NOT}(c(ID)).[a(ID) \text{ SEQ } b(ID)]. \]

Figure: Throughput (a) Various operators (b) Negation
Tests III: Stock price change on a real data set

stockIncr(ID, Adj1, Adj2) ←
stock(ID, Date1, Opn1, High1, Low1, Cls1, Vol1, Adj1)
SEQ
stock(ID, Date2, Opn2, High2, Low2, Cls2, Vol2, Adj2)
WHERE (Adj1 * X < Adj2).

- Yahoo Finance: IBM stocks from 1962 up to now
- 5% revision tuples introduced
Agenda

- Introduction, Motivation

- ETALIS Language for Events
  - Syntax;
  - Semantics;
  - Experimental Results;

Conclusion.
Conclusion: A Common Framework for Event Processing in ETALIS

ETALIS

A deductive rule framework for CEP
Integration with the domain knowledge and databases
Reasoning over streaming data and background knowledge
Retractable CEP - an extensible framework for CEP

ETALIS: A Common Framework for Event Processing
ETALIS

Open source:

http://code.google.com/p/etalis

Darko.Anicic@fzi.de