A Declarative Framework for Matching Iterative and Aggregative Patterns against Event Streams

RuleML 2011

Barcelona, Spain

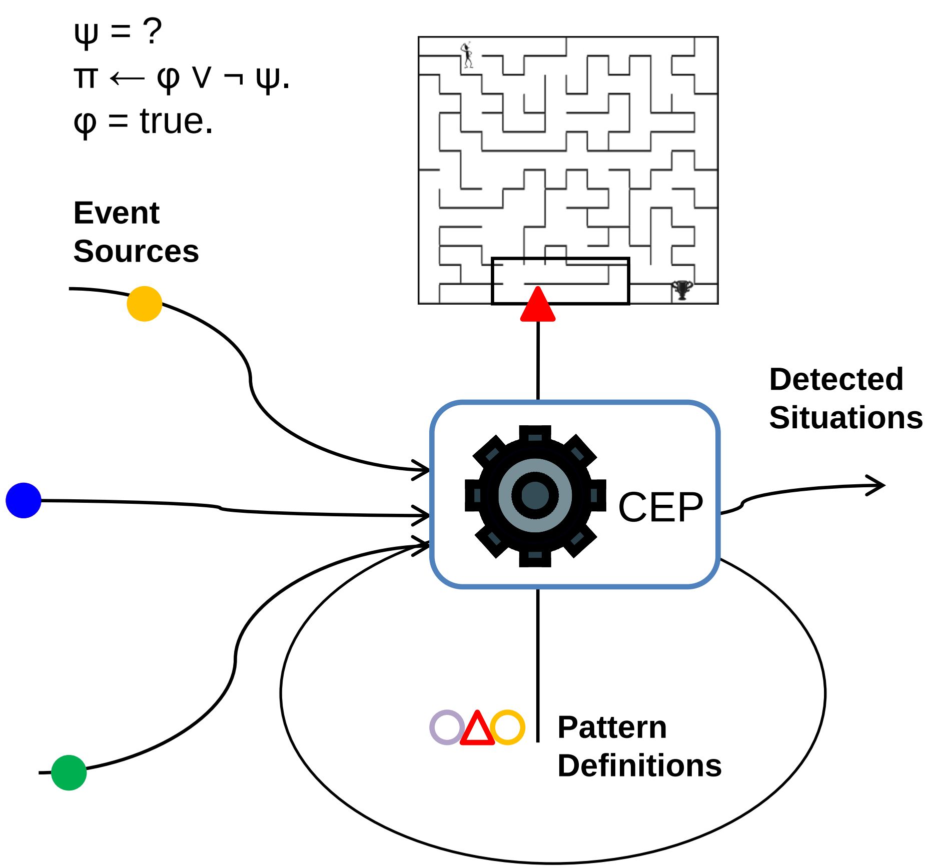
Darko Anicic,
Sebastian Rudolph,
Paul Fodor,
Nenad Stojanovic

Agenda

- Introduction, Motivation
- Iterative and Aggregative Patterns in ETALIS
  - Language
  - Window operations
  - Evaluation results
- Conclusion.

Iterative and Aggregative Patterns in ETALIS

Recursive CEP in ETALIS



ETALIS Features:

- Logic-based CEP
- Knowledge-based CEP
- Stream (deductive) reasoning
- Iterative and aggregative patterns
- An implementation available
- An extensible framework

Motivation

- In CEP it is common to perform various **aggregations** over data carried by events;
- **Iterative patterns**: an output (complex) event is treated as an input event of the same CEP processing agent;
- **Kleene closure** can be used to extract from the input stream a finite yet unbounded number of events with a particular property;
- Iterative and aggregative patterns, as well as Kleene closure can be realised with **NFA**s [Agrawal et al. SIGMOD’08] or **event graphs** [Mei et al. SIGMOD’09];
- **ETALIS** is a **logic rule-based** approach for CEP.

ETALIS - A Logic Rule-based Approach for CEP

- A rule-based formalism is **expressive** enough and convenient to represent diverse complex event patterns;
- Rules can easily express complex relationships between events by matching certain **temporal**, **relational** or **causal** conditions;
- Our rule-based formalism can specify and evaluate **contextual knowledge** (which captures the domain of interest or context related to business critical actions);
- With background knowledge events can be: **enriched**, **classified**, **clustered**, **filtered** etc. (with no **explicit** specifications).
- ETALIS is a **unified** framework for **CEP** and **reasoning**.

ETALIS: Language Syntax

ETALIS Language for Events is formally defined by:

$$\begin{aligned} P ::= \mathtt{pr}(t_1, \ldots, t_n) \quad & \mid P \text{ WHERE } t \mid q \mid (P).q \\ & \mid P \text{ BIN } P \mid \text{NOT}(P).[P, P] \end{aligned}$$

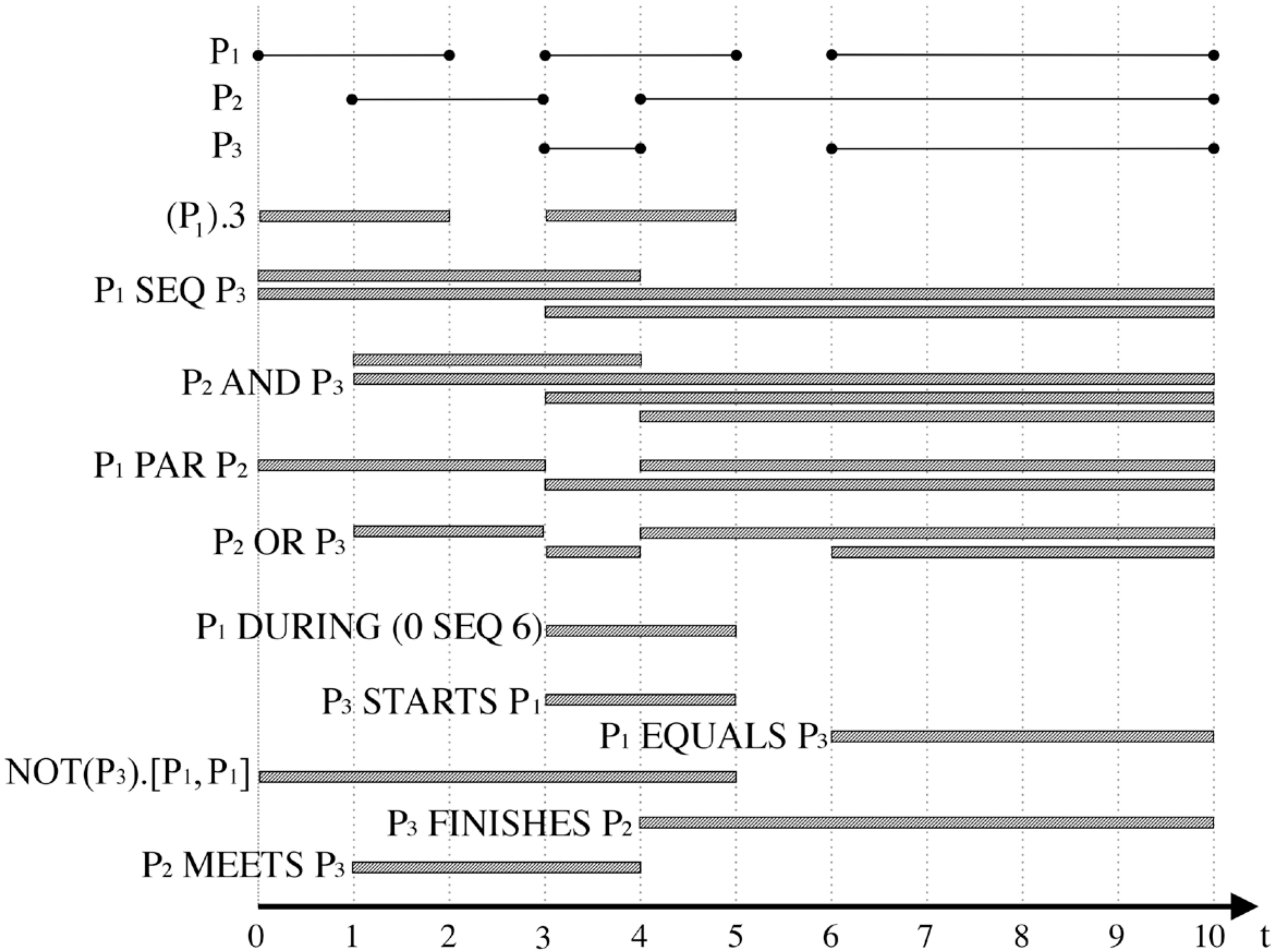
- pr - a predicate name with arity n;
- $t_{(i)}$ - denote terms;
- t - is a term of type boolean;
- q - is a nonnegative rational number;
- BIN - is one of the binary operators: SEQ, AND, PAR, OR, EQUALS, MEETS, STARTS, or FINISHES.

Event rule is defined as a formula of the following shape:

$$\mathtt{pr}(t_1, \ldots, t_n) \leftarrow p$$

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

ETALIS: Interval-based Semantics



ETALIS: Declarative Semantics

pattern	$\mathcal{I}_\mu$(pattern)
$\mathtt{pr}(t_1,\ldots,t_n)$	$\mathcal{I}(\mathtt{pr}(\mu^*(t_1),\ldots,\mu^*(t_n)))$
p WHERE t	$\mathcal{I}_\mu(p)$ if $\mu^*(t) = \mathit{true}$ $\emptyset$ otherwise.
q	$\{\langle q,q\rangle\}$ for all $q \in \mathbb{Q}^+$
$(p).q$	$\mathcal{I}_\mu(p) \cap \{\langle q_1,q_2\rangle \mid q_2 - q_1 = q\}$
p_1 SEQ p_2	$\{\langle q_1,q_4\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2,q_3 \in \mathbb{Q}^+$ with $q_2 < q_3\}$
p_1 AND p_2	$\{\langle \min(q_1,q_3),\max(q_2,q_4)\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2,q_3 \in \mathbb{Q}^+\}$
p_1 PAR p_2	$\{\langle \min(q_1,q_3),\max(q_2,q_4)\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2,q_3 \in \mathbb{Q}^+$ with $\max(q_1,q_3) < \min(q_2,q_4)\}$
p_1 OR p_2	$\mathcal{I}_\mu(p_1) \cup \mathcal{I}_\mu(p_2)$
p_1 EQUALS p_2	$\mathcal{I}_\mu(p_1) \cap \mathcal{I}_\mu(p_2)$
p_1 MEETS p_2	$\{\langle q_1,q_3\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_2,q_3\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2 \in \mathbb{Q}^+\}$
p_1 DURING p_2	$\{\langle q_3,q_4\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2,q_3 \in \mathbb{Q}^+$ with $q_3 < q_1 < q_2 < q_4\}$
p_1 STARTS p_2	$\{\langle q_1,q_3\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_1,q_3\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2 \in \mathbb{Q}^+$ with $q_2 < q_3\}$
p_1 FINISHES p_2	$\{\langle q_1,q_3\rangle \mid \langle q_2,q_3\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_1,q_3\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2 \in \mathbb{Q}^+$ with $q_1 < q_2\}$
NOT$(p_1).[p_2,p_3]$	$\mathcal{I}_\mu(p_2$ SEQ $p_3) \setminus \mathcal{I}_\mu(p_2$ SEQ p_1 SEQ $p_3)$

Definition of extensional interpretation of event patterns. We use $p_{(x)}$ for patterns, $q_{(x)}$ for rational numbers, $t_{(x)}$ for terms and $\mathtt{pr}$ for event predicates.

Examples

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

$$\begin{aligned} \mathtt{income}(Price) &\leftarrow \mathtt{sell}(Item, Price). \\ \mathtt{income}(P1 + P2) &\leftarrow \mathtt{income}(P1)\ \textsc{seq}\ \mathtt{sell}(Item, P2). \\ \mathtt{bigincome} &\leftarrow \mathtt{income}(Price)\ \textsc{where}\ Price > 100000. \end{aligned}$$

The k-fold sequential execution of an event a:

$$\begin{aligned} \mathtt{iteration}(a, 1) &\leftarrow a. \\ \mathtt{iteration}(a, k+1) &\leftarrow a\ \textsc{seq}\ \mathtt{iteration}(a, k). \end{aligned}$$

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

$$\begin{aligned} \mathtt{iteration}(a, 1) &\leftarrow a. \\ \mathtt{iteration}(a, k+1) &\leftarrow \textsc{not}(a).[a, \mathtt{iteration}(a, k)]. \\ \mathtt{e} &\leftarrow \mathtt{iteration}(a, n). \end{aligned}$$

ETALIS: Operational Semantics - AND (1/6)

1. Complex pattern (not event-driven rule)

a SEQ b SEQ c ⟶ ce1

2. Decoupling

((a SEQ b) SEQ c) ⟶ ce1

3. Binarization

a SEQ b ⟶ ie

ie SEQ c ⟶ ce1

4. Event-driven backward chaining rules

Algorithm 2 Conjunction.

Input: event binary goal $\mathtt{ie} \leftarrow \mathtt{a}\ \ \mathrm{AND}\ \mathtt{b}$.

Output: event-driven backward chaining rules for AND operator.

Each event binary goal $\mathtt{ie} \leftarrow \mathtt{a}\ \ \mathrm{AND}\ \mathtt{b}$ is converted into: {

$$\begin{array}{rl}
\mathtt{a}(T_1,T_2) :- & \mathtt{for_each}(\mathtt{a},1,[T_1,T_2]). \\
\mathtt{a}(1,T_3,T_4) :- & \mathtt{goal}(\mathtt{a}(_,_),\mathtt{b}(T_1,T_2),\mathtt{ie}(_,_)), \\
& \mathtt{retract}(\mathtt{goal}(\mathtt{a}(_,_),\mathtt{b}(T_1,T_2),\mathtt{ie}(_,_))), \\
& T_5 = \mathtt{min}\{T_1,T_3\}, T_6 = \mathtt{max}\{T_2,T_4\}, \mathtt{ie}(T_5,T_6). \\
\mathtt{a}(2,T_3,T_4) :- & \neg(\mathtt{goal}(\mathtt{a}(_,_),\mathtt{b}(T_1,T_2),\mathtt{ie}(_,_))), \\
& \mathtt{assert}(\mathtt{goal}(\mathtt{b}(_,_),\mathtt{a}(T_3,T_4),\mathtt{ie}(_,_))). \\
\mathtt{b}(T_1,T_2) :- & \mathtt{for_each}(\mathtt{b},1,[T_1,T_2]). \\
\mathtt{b}(1,T_3,T_4) :- & \mathtt{goal}(\mathtt{b}(_,_),\mathtt{a}(T_1,T_2),\mathtt{ie}(_,_)), \\
& \mathtt{retract}(\mathtt{goal}(\mathtt{b}(_,_),\mathtt{a}(T_1,T_2),\mathtt{ie}(_,_))), \\
& T_5 = \mathtt{min}\{T_1,T_3\}, T_6 = \mathtt{max}\{T_2,T_4\}, IE(T_5,T_6). \\
\mathtt{b}(2,T_3,T_4) :- & \neg(\mathtt{goal}(\mathtt{b}(_,_),\mathtt{a}(T_1,T_2),\mathtt{ie}(_,_))), \\
& \mathtt{assert}(\mathtt{goal}(\mathtt{a}(_,_),\mathtt{b}(T_3,T_4),\mathtt{ie}(_,_))).
\end{array}$$

}

ETALIS: Operational Semantics (3/6)

For any aggregate function, calculated over a window, we implement the following three rules:

$$
\begin{aligned}
&\texttt{iteration}(StartCntr = 0, StartVal) \leftarrow \texttt{start\_event}(StartVal).\\
&\texttt{iteration}(OldCntr + 1, NewVal) \leftarrow\\
&\quad\quad \texttt{iteration}(OldCntr, OldVal)\ \textsc{seq}\ \mathsf{a}(AggArg)\\
&\quad\quad\ \textsc{where}\ \{\texttt{assert}(AggArg),\\
&\quad\quad\quad\quad \texttt{window}(WndwSize, OldCntr, OldVal, AggArg, NewVal)\}.
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{window}(WndwSize, OldCntr, OldVal, AggArg, NewVal) : -\\
&\quad\quad OldCntr + 1 >= WindowSize - >\\
&\quad\quad \texttt{retract}(LastItem),\\
&\quad\quad \texttt{spec\_aggregate}(OldValue, AggArg, NewValue);\\
&\quad\quad \texttt{spec\_aggregate}(OldValue, AggArg, NewValue).
\end{aligned}
$$

ETALIS: Operational Semantics (4/6)

SUM aggregate function:

$$\texttt{sum}(StartCntr = 0, StartVal) \leftarrow \texttt{start\_event}(StartVal).$$

$$\begin{aligned}
&\texttt{sum}(OldCntr + 1, NewSum) \leftarrow \\
&\quad \texttt{sum}(OldCntr + 1, OldSum)\ \textsf{SEQ}\ \textsf{a}(AggArg) \\
&\quad \textsf{WHERE}\ \{\texttt{assert}(AggArg), \\
&\qquad \texttt{window}(WndwSize, OldCntr, \\
&\qquad OldSum + AggArg, AggArg, NewSum)\}.
\end{aligned}$$

$$\begin{aligned}
&\texttt{window}(WndwSize, OldCntr, CurrSum, NewSum) :- \\
&\quad OldCntr + 1 >= WindowSize -> \\
&\quad \texttt{retract}(LastItem), \\
&\quad NewSum = CurrSum - LastItem; \\
&\quad NewSum = CurrSum - LastItem.
\end{aligned}$$

ETALIS: Operational Semantics (5/6)

MAX aggregate function:

$$
\begin{aligned}
&\mathtt{max}(StartCntr = 0, StartVal) \leftarrow \mathtt{start\_event}(StartVal).\\
&\mathtt{max}(OldCntr + 1, NewMax) \leftarrow\\
&\qquad \mathtt{max}(OldCntr + 1, OldMax)\ \mathrm{SEQ}\ \mathtt{a}(AggArg)\\
&\qquad \ \mathrm{WHERE}\ \{\mathtt{assert}(AggArg),\\
&\qquad\qquad \mathtt{window}(WndwSize, OldCntr, NewMax)\}.
\end{aligned}
$$

$$
\begin{aligned}
&\mathtt{window}(WndwSize, OldCntr, NewMax) :-\\
&\qquad OldCntr + 1 >= WindowSize- >\\
&\qquad \mathtt{retract}(LastItem), get(NewMax);\\
&\qquad get(NewMax).
\end{aligned}
$$

ETALIS: Operational Semantics (6/6)

COUNT aggregate function:

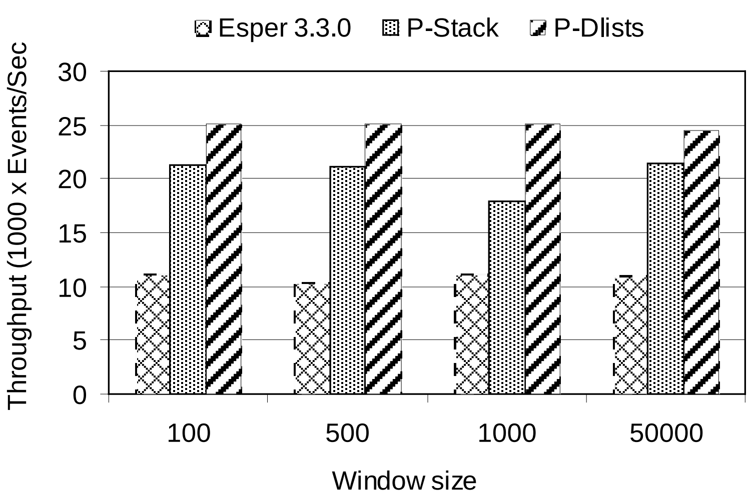
$$\texttt{iteration}(StartCntr = 0, StartVal) \leftarrow \texttt{start\_event}(StartVal).$$
$$\texttt{iteration}(NewCntr) \leftarrow$$
$$\quad \texttt{iteration}(OldCntr) \text{ SEQ } \texttt{a}(AggArg)$$
$$\quad \text{WHERE } \{NewCntr = \texttt{getCount}([T_2, T_1]), \texttt{window}(3min)\}.$$

- 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

Tests I: Throughput Comparison

- 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
- Aggregations are computed over complex events of the following type:

$$\mathsf{a}(ID, X, Y) \leftarrow \mathsf{b}(ID, X) \text{ AND } \mathsf{c}(ID, Y).$$



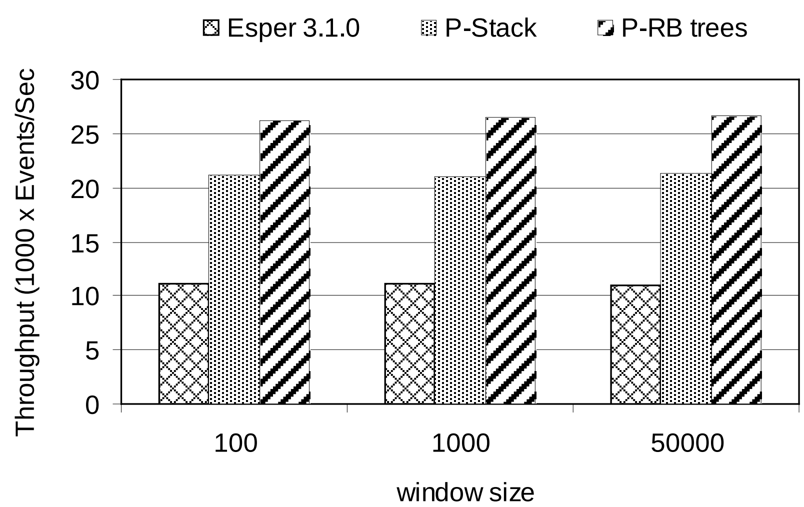


Figure: Throughput vs. wind. size (a) SUM-AND (b) AVG-SEQ

Tests II: CEP with Stream Reasoning

$$\texttt{delivery}(start, start) \leftarrow \texttt{shipment}(start).$$
$$\texttt{delivery}(From, To) \leftarrow \texttt{delivery}(From, PrevTo)$$
$$\textsc{seq}\ \texttt{shipment}(To)$$
$$\textsc{where}\ \texttt{inSupChain}(From, To).$$

$$\texttt{inSupChain}(X, Y) :- \texttt{linked}(X, Y).$$
$$\texttt{inSupChain}(X, Z) :- \texttt{linked}(X, Y)\ \textsc{and}\ \texttt{inSupChain}(Y, Z).$$

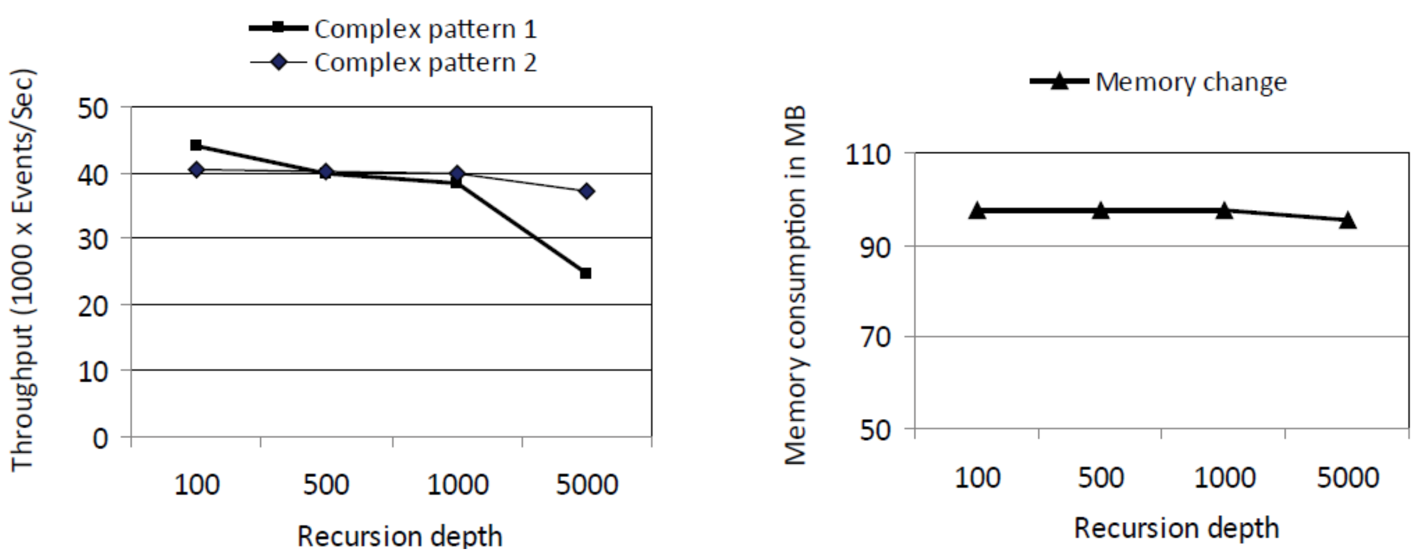


Figure: (a) throughput comparison (b) memory consumption

Conclusion

- We presented a framework for a logic rule-based CEP and reasoning;
- The framework is capable to specify and process iterative and aggregative complex event patterns;
- Aggregations, calculated over sliding windows, do not require an extension of the existing language and come with no additional costs in terms of performance.

Thank you! Questions…

ETALIS

Open source:

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

Darko.Anicic@fzi.de