Complex Event Recognition in Multi-Agent Systems

Alexander Artikis\textsuperscript{1,2} & Matthias Weidlich\textsuperscript{3}

\textsuperscript{1}University of Piraeus, Greece
\textsuperscript{2}National Centre for Scientific Research “Demokritos”, Greece
\textsuperscript{3}Imperial College London, UK

a.artikis@unipi.gr, m.weidlich@imperial.ac.uk
Complex Event Recognition (Event Pattern Matching)

Input:

- Symbolic representation of time-stamped, simple, derived events (SDE) coming from (geographically distributed) sources.
- Big Data.

Output:

- Complex or composite events (CE) — collections of SDE and/or CE that satisfy some pattern (temporal, spatial, logical constraints).
  - Not restricted to aggregates.
- Humans understand CE easier than SDE.
Complex Event Recognition

INPUT ▶ RECOGNITION ▶ OUTPUT ■

Event Recognition System

CE Definitions

Streams of SDEs

Recognised CEs
Example: Complex Event Recognition for Security
<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>340 inactive($id_0$)</td>
<td></td>
</tr>
<tr>
<td>340 $p(id_0) = (20.88, -11.90)$</td>
<td></td>
</tr>
<tr>
<td>340 appear($id_0$)</td>
<td></td>
</tr>
<tr>
<td>340 walking($id_2$)</td>
<td></td>
</tr>
<tr>
<td>340 $p(id_2) = (25.88, -19.80)$</td>
<td></td>
</tr>
<tr>
<td>340 active($id_1$)</td>
<td></td>
</tr>
<tr>
<td>340 $p(id_1) = (20.88, -11.90)$</td>
<td></td>
</tr>
<tr>
<td>340 walking($id_3$)</td>
<td></td>
</tr>
<tr>
<td>340 $p(id_3) = (24.78, -18.77)$</td>
<td></td>
</tr>
<tr>
<td>380 walking($id_3$)</td>
<td></td>
</tr>
<tr>
<td>380 $p(id_3) = (27.88, -9.90)$</td>
<td></td>
</tr>
<tr>
<td>380 walking($id_2$)</td>
<td></td>
</tr>
<tr>
<td>380 $p(id_2) = (28.27, -9.66)$</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>340  \textit{inactive}(id_0)</td>
<td>340 \textit{leaving object}(id_1, id_0)</td>
</tr>
<tr>
<td>340 \textit{p}(id_0) = (20.88, -11.90)</td>
<td></td>
</tr>
<tr>
<td>340 \textit{appear}(id_0)</td>
<td></td>
</tr>
<tr>
<td>340 \textit{walking}(id_2)</td>
<td></td>
</tr>
<tr>
<td>340 \textit{p}(id_2) = (25.88, -19.80)</td>
<td></td>
</tr>
<tr>
<td>340 \textit{active}(id_1)</td>
<td></td>
</tr>
<tr>
<td>340 \textit{p}(id_1) = (20.88, -11.90)</td>
<td></td>
</tr>
<tr>
<td>340 \textit{walking}(id_3)</td>
<td></td>
</tr>
<tr>
<td>340 \textit{p}(id_3) = (24.78, -18.77)</td>
<td></td>
</tr>
<tr>
<td>380 \textit{walking}(id_3)</td>
<td></td>
</tr>
<tr>
<td>380 \textit{p}(id_3) = (27.88, -9.90)</td>
<td></td>
</tr>
<tr>
<td>380 \textit{walking}(id_2)</td>
<td></td>
</tr>
<tr>
<td>380 \textit{p}(id_2) = (28.27, -9.66)</td>
<td></td>
</tr>
</tbody>
</table>
### Complex Event Recognition for Security

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>340 inactive$(id_0)$</td>
<td>340 leaving_object$(id_1, id_0)$</td>
</tr>
<tr>
<td>340 $p(id_0) = (20.88, -11.90)$</td>
<td>since(340) moving$(id_2, id_3)$</td>
</tr>
<tr>
<td>340 appear$(id_0)$</td>
<td></td>
</tr>
<tr>
<td>340 walking$(id_2)$</td>
<td></td>
</tr>
<tr>
<td>340 $p(id_2) = (25.88, -19.80)$</td>
<td></td>
</tr>
<tr>
<td>340 active$(id_1)$</td>
<td></td>
</tr>
<tr>
<td>340 $p(id_1) = (20.88, -11.90)$</td>
<td></td>
</tr>
<tr>
<td>340 walking$(id_3)$</td>
<td></td>
</tr>
<tr>
<td>340 $p(id_3) = (24.78, -18.77)$</td>
<td></td>
</tr>
<tr>
<td>380 walking$(id_3)$</td>
<td></td>
</tr>
<tr>
<td>380 $p(id_3) = (27.88, -9.90)$</td>
<td></td>
</tr>
<tr>
<td>380 walking$(id_2)$</td>
<td></td>
</tr>
<tr>
<td>380 $p(id_2) = (28.27, -9.66)$</td>
<td></td>
</tr>
</tbody>
</table>
Complex Event Recognition for Security

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>420 active(id₄)</td>
<td></td>
</tr>
<tr>
<td>420 p(id₄) = (10.88, -71.90)</td>
<td></td>
</tr>
<tr>
<td>420 inactive(id₃)</td>
<td></td>
</tr>
<tr>
<td>420 p(id₃) = (5.8, -50.90)</td>
<td></td>
</tr>
<tr>
<td>420 abrupt(id₅)</td>
<td></td>
</tr>
<tr>
<td>420 p(id₅) = (11.80, -72.80)</td>
<td></td>
</tr>
<tr>
<td>420 active(id₆)</td>
<td></td>
</tr>
<tr>
<td>420 p(id₆) = (7.8, -52.90)</td>
<td></td>
</tr>
<tr>
<td>480 abrupt(id₄)</td>
<td></td>
</tr>
<tr>
<td>480 p(id₄) = (20.45, -12.90)</td>
<td></td>
</tr>
<tr>
<td>480 abrupt(id₅)</td>
<td></td>
</tr>
<tr>
<td>480 p(id₅) = (17.88, -11.90)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
## Complex Event Recognition for Security

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>420 active($id_4$)</td>
<td>[420, 480] fighting($id_4$, $id_5$)</td>
</tr>
<tr>
<td>420 $p(id_4)$ = (10.88, -71.90)</td>
<td></td>
</tr>
<tr>
<td>420 inactive($id_3$)</td>
<td></td>
</tr>
<tr>
<td>420 $p(id_3)$ = (5.8, -50.90)</td>
<td></td>
</tr>
<tr>
<td>420 abrupt($id_5$)</td>
<td></td>
</tr>
<tr>
<td>420 $p(id_5)$ = (11.80, -72.80)</td>
<td></td>
</tr>
<tr>
<td>420 active($id_6$)</td>
<td></td>
</tr>
<tr>
<td>420 $p(id_6)$ = (7.8, -52.90)</td>
<td></td>
</tr>
<tr>
<td>480 abrupt($id_4$)</td>
<td></td>
</tr>
<tr>
<td>480 $p(id_4)$ = (20.45, -12.90)</td>
<td></td>
</tr>
<tr>
<td>480 abrupt($id_5$)</td>
<td></td>
</tr>
<tr>
<td>480 $p(id_5)$ = (17.88, -11.90)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Example: City Transport & Traffic Management
## City Transport & Traffic Management

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>scheduled stop enter</td>
</tr>
<tr>
<td>215</td>
<td>late stop leave</td>
</tr>
<tr>
<td>[215, 400]</td>
<td>abrupt acceleration</td>
</tr>
<tr>
<td>[350, 600]</td>
<td>sharp turn</td>
</tr>
<tr>
<td>620</td>
<td>flow=low</td>
</tr>
<tr>
<td></td>
<td>density=high</td>
</tr>
<tr>
<td>700</td>
<td>scheduled stop enter</td>
</tr>
<tr>
<td>720</td>
<td>flow=low</td>
</tr>
<tr>
<td></td>
<td>density=average</td>
</tr>
<tr>
<td>820</td>
<td>scheduled stop leave</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
## City Transport & Traffic Management

<table>
<thead>
<tr>
<th><strong>Input</strong></th>
<th><strong>Output</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>scheduled stop enter</td>
</tr>
<tr>
<td>215</td>
<td>late stop leave</td>
</tr>
<tr>
<td>[215, 400]</td>
<td>abrupt acceleration</td>
</tr>
<tr>
<td>[350, 600]</td>
<td>sharp turn</td>
</tr>
<tr>
<td>620</td>
<td><em>flow</em> = low</td>
</tr>
<tr>
<td></td>
<td><em>density</em> = high</td>
</tr>
<tr>
<td>700</td>
<td>scheduled stop enter</td>
</tr>
<tr>
<td>720</td>
<td><em>flow</em> = low</td>
</tr>
<tr>
<td></td>
<td><em>density</em> = average</td>
</tr>
<tr>
<td>820</td>
<td>scheduled stop leave</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
### City Transport & Traffic Management

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>scheduled stop enter</td>
</tr>
</tbody>
</table>
| 215     | late stop leave  
|         | since(215) non-punctual |
| [215, 400] | abrupt acceleration |
| [350, 600] | sharp turn  
|         | [215, 600] uncomfortable driving |
| 620     | flow=low  
|         | density=high  
|         | since(620) congestion |
| 700     | scheduled stop enter |
| 720     | flow=low  
|         | density=average  
|         | [620, 720] congestion |
| 820     | scheduled stop leave  
|         | [215, 820] non-punctual |
| ...     | ...     |
Other Applications

- Maritime surveillance.
- Energy management.
- Cardiac arrhythmia detection.
- Humpback whale song recognition.
- Credit card fraud recognition.
Complex Event Recognition Research

- Efficient reasoning
  - to support real-time decision-making in large-scale, (geographically) distributed applications.
- Reasoning under uncertainty
  - to deal with various types of noise.
- Automated knowledge construction
  - to avoid the time-consuming, error-prone manual CE definition development.
Complex Event Recognition in Multi-Agent Systems

Detect in real-time composite activities & situations of ‘special significance’ within a (very large) MAS:

▶ Normative relations
  ▶ Institutional power, permission, obligation, commitment, entitlement, authorisation, right, etc.

▶ Non-compliant behaviour
  ▶ Unavoidable non-compliance (an agent fails to behave as intended because of factors beyond its control).
  ▶ Malicious behaviour (an agent could have complied with norms but did not).

▶ Agent & team strategies
  ▶ Robocup soccer simulation, robotic urban search & rescue operations, unmanned expeditions, etc.

▶ Agent & system performance
  ▶ Voting procedure manipulated by strategic voting, inefficient resource allocation (in energy markets), etc.
Complex Event Recognition in Multi-Agent Systems

Event Recognition System

Activity Definitions

Agent messages & observations

Recognised Activities
Tutorial Structure

- Temporal reasoning systems.
- Optimisation.
- Open issues.
Tutorial Structure

- Temporal reasoning systems.
- Optimisation.
- Open issues.
Composite Event Algebra

Core components of an event algebra with point-based semantics:

- **Sequencing** (SEQ) lists the required event types in temporal order — eg \(\text{SEQ}(A, B, C)\).

- **Kleene closure** (+) collects a finite yet unbound number of events of a particular type. It is used as a component of SEQ — eg \(\text{SEQ}(A, B+, C)\).

- **Negation** (\(\sim\) or !) verifies the absence of certain events in a sequence — eg \(\text{SEQ}(A, !B, C)\).

- **Value predicates** specify constraints on the event attributes
  - Aggregate functions \(\text{max}, \text{min}, \text{count}, \text{sum}, \text{avg}\).
Composite Event Algebra

- **Composition** refers to:
  - **Union** of constraints — eg $\text{SEQ}(A, B, C) \cup \text{SEQ}(A, D, E)$.
  - **Negation of a sequence** — eg $\neg\text{SEQ}(A, B, C)$.
  - **Kleene closure of a constraint** — eg $\text{SEQ}(A, B, C)^+$.

- **Windowing** (WITHIN) restricts a CE definition to a specific time period.

- **Output construction** (RETURN) defines the events to emit upon satisfying the CE definition.
## Chronicle Recognition System

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>event(E, T)</code></td>
<td>Event E takes place at time T</td>
</tr>
<tr>
<td><code>event(F:(?V1,?V2),T)</code></td>
<td>An event takes place at time T changing the value of property F from ?V1 to ?V2</td>
</tr>
<tr>
<td><code>noevent(E, (T1,T2))</code></td>
<td>Event E does not take place between [T1,T2)</td>
</tr>
<tr>
<td><code>noevent(F:(?V1,?V2), (T1,T2))</code></td>
<td>No event takes place between [T1,T2) that changes the value of property F from ?V1 to ?V2</td>
</tr>
<tr>
<td><code>hold(F:?V, (T1,T2))</code></td>
<td>The value of property F is ?V between [T1,T2)</td>
</tr>
<tr>
<td><code>occurs(N,M,E,(T1,T2))</code></td>
<td>Event E takes place at least N times and at most M times between [T1,T2)</td>
</tr>
</tbody>
</table>
chronicle contract[?c, ?m, ?item, ?price](T2) {
    event( request_quote[?c, ?m, ?item], T0 )
    event( present_quote[?m, ?c, ?item, ?price], T1 )
    event( accept_quote[?c, ?m, ?item, ?price], T2 )
    T1 > T0
    T2 > T1
    end - start in [1, 2000]
}

chronicle contract_violation[?m, ?item, ?price](T0+10000) {
    event( contact[?c, ?m, ?item, ?price], T0 )
    noevent( deliver[?m, ?c, ?item], ( T0+1, T0+10000 ) )
}
Chronicle Representation Language

- Mathematical operators in the atemporal constraints of the language are not allowed
- Universal quantification is not allowed

CRS is a purely temporal reasoning system.

It is also a very efficient and scalable system.
Chronicle Recognition System: Semantics

Each CE definition is represented as a Temporal Constraint Network. Eg:

present_quote \([?m,?c,?item,?price]\) 

\[1,1000]\] 

accept_quote \([?c,?m,?item,?price]\)
Chronicle Recognition System: Consistency Checking

Compilation stage:
- Constraint propagation in the Temporal Constraint Network.
- Consistency checking.
Chronicle Recognition System: Recognition

Recognition stage:

- Partial CE instance evolution.
- Forward (predictive) recognition.
CE definition: contract between two parties

A: request quote
B: present quote
C: accept quote
CE definition: contract between two parties

A: request quote
B: present quote
C: accept quote
Chronicle Recognition System: Partial instances

CE definition: contract between two parties

A: request quote
B: present quote
C: accept quote

A@1

B[2,4]

A@1

B[2,4]
CE definition: contract between two parties

A: request quote
B: present quote
C: accept quote
Chronicle Recognition System: Partial instances

CE definition: contract between two parties

A: request quote
B: present quote
C: accept quote

A@1 → B[2,4] → C

A@1 → B[4,4]
A@3 → B[4,6]
CE definition: contract between two parties

A: request quote
B: present quote
C: accept quote
CE definition: contract between two parties

A: request quote
B: present quote
C: accept quote

A@1 \rightarrow B[2,4]
A@3 \rightarrow B[4,4]
A@3 \rightarrow B[4,6]
B@5 \rightarrow time
Chronicle Recognition System: Partial instances

CE definition: contract between two parties

A: request quote
B: present quote
C: accept quote

A@1 $\rightarrow$ B[2,4] $\rightarrow$ B[4,4] $\rightarrow$ C[5,8]

killed instance

A@3 $\rightarrow$ B[4,6] $\rightarrow$ C[5,8]
Chronicle Recognition System: Partial instances

CE definition: contract between two parties

A: request quote
B: present quote
C: accept quote

A@1 → B[2,4]  
A@1 → B[4,4]  
A@3 → B[4,6]  
A@3 → B[5,6]  
A@3 → B[5,6]  
A@3 → B[5,8]  
B@5  
C[5,8]  
killed instance

duplicated

time
Recognition stage — partial CE instance management:

- In order to manage all the partial CE instances, CRS stores them in trees, one for each CE definition.
- Each event occurrence and each clock tick traverses these trees in order to kill some CE instances (tree nodes) or to develop some CE instances.
- For $K$ CE instances, each having $n$ subevents, the complexity of processing each incoming event or a clock update is $O(Kn^2)$.
- To deal with out-of-order SDE streams, CRS keeps in memory partial CE instances longer.
Chronicle Recognition System: Summary

- One of the earliest and most successful formal complex event recognition systems.
- Being AI-based, it has been largely overlooked by the event processing community.
- Very efficient and scalable event recognition.
- **But:**
  - It is a purely temporal reasoning system.
  - It does not handle uncertainty.
Event Calculus

- A logic programming language for representing and reasoning about events and their effects.
- Key components:
  - event (typically instantaneous).
  - fluent: a property that may have different values at different points in time.
- Built-in representation of **inertia**: 
  - $F = V$ holds at a particular time-point if $F = V$ has been *initiated* by an event at some earlier time-point, and not *terminated* by another event in the meantime.
## Run-Time Event Calculus

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>happensAt((E, T))</td>
<td>Event (E) occurs at time (T)</td>
</tr>
<tr>
<td>initiatedAt((F = V, T))</td>
<td>At time (T) a period of time for which (F = V) is initiated</td>
</tr>
<tr>
<td>terminatedAt((F = V, T))</td>
<td>At time (T) a period of time for which (F = V) is terminated</td>
</tr>
<tr>
<td>holdsFor((F = V, I))</td>
<td>(I) is the list of the maximal intervals for which (F = V) holds continuously</td>
</tr>
<tr>
<td>holdsAt((F = V, T))</td>
<td>The value of fluent (F) is (V) at time (T)</td>
</tr>
<tr>
<td>union_all([J_1, \ldots, J_n], I)</td>
<td>(I = (J_1 \cup \ldots \cup J_n))</td>
</tr>
<tr>
<td>intersect_all([J_1, \ldots, J_n], I)</td>
<td>(I = (J_1 \cap \ldots \cap J_n))</td>
</tr>
<tr>
<td>relative_complement_all((I', [J_1, \ldots, J_n], I))</td>
<td>(I = I' \setminus (J_1 \cup \ldots \cup J_n))</td>
</tr>
</tbody>
</table>
CE Definitions in RTEC: Simple Fluents

**CE definition:**

\[\text{initiatedAt}(\text{CE}, T) \leftarrow \text{happensAt}(E_{In_1}, T), \text{conditions}\]
\[\ldots\]
\[\text{initiatedAt}(\text{CE}, T) \leftarrow \text{happensAt}(E_{In_i}, T), \text{conditions}\]

where

**conditions:**

\[0^{-K}\text{happensAt}(E_k, T),\]
\[0^{-M}\text{holdsAt}(F_m, T),\]
\[0^{-N}\text{atemporal-constraint}_n\]
CE Definitions in RTEC: Simple Fluents

CE definition:

\[
\text{initiatedAt}(CE, T) \leftarrow \text{happensAt}(E_{I_n}, T), \quad \text{[conditions]}
\]

\[
\ldots
\]

\[
\text{initiatedAt}(CE, T) \leftarrow \text{happensAt}(E_{I_n}, T), \quad \text{[conditions]}
\]

\[
\text{terminatedAt}(CE, T) \leftarrow \text{happensAt}(E_{T_1}, T), \quad \text{[conditions]}
\]

\[
\ldots
\]

\[
\text{terminatedAt}(CE, T) \leftarrow \text{happensAt}(E_{T_j}, T), \quad \text{[conditions]}
\]

CE recognition:
CE Definitions in RTEC: Simple Fluents

CE definition:

\[
\begin{align*}
\text{initiatedAt}(CE, T) & \leftarrow \text{happensAt}(E_{In_i}, T), \\
\hspace{1cm} & \hspace{1cm} \text{[conditions]} \\
\ldots \hspace{1cm} & \hspace{1cm} \ldots \\
\text{initiatedAt}(CE, T) & \leftarrow \text{happensAt}(E_{In_i}, T), \\
\hspace{1cm} & \hspace{1cm} \text{[conditions]} \\
\text{terminatedAt}(CE, T) & \leftarrow \text{happensAt}(E_{Tj}, T), \\
\hspace{1cm} & \hspace{1cm} \text{[conditions]}
\end{align*}
\]

CE recognition:

\[
\begin{align*}
\text{time} & \hspace{1cm} \text{time}
\end{align*}
\]
CE Definitions in RTEC: Simple Fluents

CE definition:

\[
\text{initiatedAt}(CE, T) \leftarrow \\
\text{happensAt}(E_{ln_1}, T), \\ 
\text{conditions}
\]

\[
\text{...}
\]

\[
\text{initiatedAt}(CE, T) \leftarrow \\
\text{happensAt}(E_{ln_i}, T), \\ 
\text{conditions}
\]

\[
\text{terminatedAt}(CE, T) \leftarrow \\
\text{happensAt}(E_{T_1}, T), \\ 
\text{conditions}
\]

\[
\text{...}
\]

\[
\text{terminatedAt}(CE, T) \leftarrow \\
\text{happensAt}(E_{T_j}, T), \\ 
\text{conditions}
\]

CE recognition:
CE Definitions in RTEC: Simple Fluents

CE definition:

\[
\text{initiatedAt}(CE, T) \leftarrow \text{happensAt}(E_{In_1}, T), \quad \text{[conditions]}
\]

\[\ldots\]

\[
\text{initiatedAt}(CE, T) \leftarrow \text{happensAt}(E_{In_i}, T), \quad \text{[conditions]}
\]

\[
\text{terminatedAt}(CE, T) \leftarrow \text{happensAt}(E_{T_1}, T), \quad \text{[conditions]}
\]

\[\ldots\]

\[
\text{terminatedAt}(CE, T) \leftarrow \text{happensAt}(E_{T_j}, T), \quad \text{[conditions]}
\]

CE recognition: \(\text{holdsFor}(CE, I)\)
CE Definitions in RTEC: Simple Fluents

CE definition:

\[
\text{initiatedAt}(\text{obl}(\text{send\_EPO}(C, M, \text{Item}, \text{Price})) = \text{true}, \quad T) \leftarrow \\
\text{initiatedAt}(\text{contract}(C, M, \text{Item}, \text{Price}) = \text{true}, \quad T) \\
\text{terminatedAt}(\text{obl}(\text{send\_EPO}(C, M, \text{Item}, \text{Price})) = \text{true}, \quad T) \leftarrow \\
\text{happensAt}(\text{send\_EPO}(C, M, \text{Item}, \text{Price}), \quad T) \\
\text{terminatedAt}(\text{obl}(\text{send\_EPO}(C, M, \text{Item}, \text{Price})) = \text{true}, \quad T) \leftarrow \\
\text{terminatedAt}(\text{contract}(C, M, \text{Item}, \text{Price}) = \text{true}, \quad T)
\]

CE recognition:

\[
\text{holdsFor}(\text{obl}(\text{send\_EPO}(C, M, \text{Item}, \text{Price})) = \text{true}, \quad I)
\]
CE Definitions in RTEC: Statically Determined Fluents

CE definition:

holdsFor(CE, I) ←
holdsFor(F_1, I_{F_1}),
...
holdsFor(F_f, I_{F_f}),
interval\_manipulation_1(I_{\alpha}, \ldots, I_{\omega}),
...
interval\_manipulation_k(I_A, \ldots, I_{\Omega})

where

interval\_manipulation(I_1, \ldots, I_n) :
\begin{align*}
I_1 \cup \cdots \cup I_n \\
I_1 \cap \cdots \cap I_n \\
I_1 \setminus I_2
\end{align*}
Interval Manipulation: Union

$I_1 \cup \cdots \cup I_n$
Interval Manipulation: Intersection

\[ I_1 \cap \cdots \cap I_n \]
Interval Manipulation: Complement

\[ I_1 \setminus I_2 \]
CE Definitions in RTEC: Statically Determined Fluents

CE definition:

\[ \text{pow}(\text{accept}_\text{quote}(C, M, Item)) \iff \text{quote}(M, C, Item), \]
\[ \text{not (suspended}(C, \text{consumer}) \text{ or suspended}(M, \text{merchant})) \]

Compiled CE definition:

\[
\text{holdsFor}(\text{pow}(\text{accept}_\text{quote}(C, M, Item)) = \text{true}, I_1 \setminus (I_2 \cup I_3)) \leftarrow \\
\text{holdsFor}(\text{quote}(M, C, Item) = \text{true}, I_1), \\
\text{holdsFor}(\text{suspended}(C, \text{consumer}) = \text{true}, I_2), \\
\text{holdsFor}(\text{suspended}(M, \text{merchant}) = \text{true}, I_3)
\]
CE Hierarchies
Run-Time Event Recognition

Real-time decision-support in the presence of:

- Very large SDE streams.
- Non-sorted SDE streams.
- SDE revision.
- Very large CE numbers.
Complex Event Recognition in RTEC
Complex Event Recognition in RTEC

![Diagram of working memory over time with events and Q177, Q178, Q179, Q180, Q181, Q182]

- Working Memory
- Time line from Q177 to Q182
Complex Event Recognition in RTEC

Diagram showing the timeline with Working Memory at different time points Q_{177} to Q_{182}.
RTEC: Summary

- Representation of complex temporal phenomena.
  - Succinct representation $\rightarrow$ code maintenance.
  - Intuitive representation $\rightarrow$ facilitates interaction with domain experts unfamiliar with programming.

- The full power of logic programming is available.
  - Complex atemporal computations in CE definitions.
  - Combination of streaming data with historical knowledge.

- Very efficient reasoning.
  - Even when SDE arrive with a delay and are revised.
  - Even in the presence of large CE hierarchies.

- But:
  - The Event Calculus does not deal with uncertainty.
Tutorial Structure

▶ Temporal reasoning systems.
▶ Optimisation.
▶ Open issues.
The Need for Optimisation

High velocity data

- Input rate of stream
- Size of each actual event
The Need for Optimisation

High velocity data
- Input rate of stream
- Size of each actual event

Applicability of event detection
- Memory consumption of partial matches may be huge
- Value of event detection may deteriorate over time
The Need for Optimisation

High velocity data
▶ Input rate of stream
▶ Size of each actual event

Applicability of event detection
▶ Memory consumption of partial matches may be huge
▶ Value of event detection may deteriorate over time

Optimisation goals
▶ Throughput, events per second
▶ Latency, time between arrival of last SDE until CE detection
▶ Typically, there is a trade-off, yet, the dimensions are not orthogonal
The Need for Optimisation Exemplified

Example setting

- Input rate of 50,000 events per second
- Each event is 100 Bytes, e.g., around 10-20 attribute values
- Query over time window of 5 minutes
The Need for Optimisation Exemplified

Example setting

- Input rate of 50,000 events per second
- Each event is 100 Bytes, e.g., around 10-20 attribute values
- Query over time window of 5 minutes

Example requirements

- Data volume per window: rate \(\times\) window size \(\times\) event size
- Here, per window: 1.5GB
- Suppose that the window slide is small, i.e., 10 seconds
- Then, in 5 minutes: 1.5GB \(\times\) 30 = 45GB
- Partial matches quickly sum up to a few GB
The Need for Optimisation Exemplified

Example setting
- Input rate of 50,000 events per second
- Each event is 100 Bytes, e.g., around 10-20 attribute values
- Query over time window of 5 minutes

Example requirements
- Data volume per window: rate \( \times \) window size \( \times \) event size
- Here, per window: 1.5GB
- Suppose that the window slide is small, i.e., 10 seconds
- Then, in 5 minutes: 1.5GB \( \times \) 30 = 45GB
- Partial matches quickly sum up to a few GB

Reality check
- Input rates, event sizes, window sizes are domain specific
- Also, impact of not meeting performance goals varies greatly
Classification of Optimisation approaches

Dimensions to classify optimisation approaches

- Single query vs. multi-query
- Language level vs. execution plan level
- Single instance vs. distributed setting
- Lossless vs. lossy optimisation
Classification of Optimisation approaches

Dimensions to classify optimisation approaches

- Single query vs. multi-query
- Language level vs. execution plan level
- Single instance vs. distributed setting
- Lossless vs. lossy optimisation

Common mechanisms include

- Control of input rate, load shedding
- Rewriting of query (high-level language) or execution plan (low-level language)
- Selection of a ‘good’ execution plan
- Indexing of queries (e.g., of similar predicates)
- Sharing of intermediate results
- Placement of query operators or execution plans
Overview of Discussed Approaches

(1) Generic rewriting
   ▶ Rewriting of automata-based plans based on cost-model
   ▶ Exploits relative costs per basic operator
Overview of Discussed Approaches

(1) **Generic rewriting**
   - Rewriting of automata-based plans based on cost-model
   - Exploits relative costs per basic operator

(2) **Postponed plan evaluation**
   - Tune the evaluation of automata-based plans to avoid replication of partial matches
   - Exploits symmetries in the matching phases for a set of CE
Overview of Discussed Approaches

(1) **Generic rewriting**
   - Rewriting of automata-based plans based on cost-model
   - Exploits relative costs per basic operator

(2) **Postponed plan evaluation**
   - Tune the evaluation of automata-based plans to avoid replication of partial matches
   - Exploits symmetries in the matching phases for a set of CE

(3) **Semantic rewriting**
   - Rewriting of high-level queries and automata-based plans using information on the event stream
   - Exploits behavioural regularities in event stream
More Details on a Automata-based Event Recognition

Composite event algebra

- Point-based operators as discussed earlier
- Evaluated over windows over stream
More Details on a Automata-based Event Recognition

Composite event algebra

► Point-based operators as discussed earlier
► Evaluated over windows over stream

Common automata-based approach to recognise CE

► Translate pattern into automata
► Automata: DFA, NFA, NFA with buffers (similar to push-down automata), ...
► Partial matches are partial runs of automata
► Further:
  ► Predicate checks over event attribute values
  ► Time window checks for partial runs
  ► Negation checks for partial runs
Example Execution Model

Execution as implemented by SASE engine
(Wu, Diao & Rizvi, SIGMOD 2006)
Generic Pattern Rewriting - Overview

Setting

- Rewriting of high-level queries and automata-based plans using information on the event stream
- Primary goal: reduce computational effort
- Secondary goal: lower detection latency

Approach by Schultz-Møller, Migliavacca & Pietzuch (DEBS 2009)

- Given high-level pattern description, select efficient automata-based plan
- Plan efficiency is quantified by cost model
- Cost model is based on arrival rates of event types
Generic Pattern Rewriting - Overview

Setting

- Rewriting of high-level queries and automata-based plans using information on the event stream
- Primary goal: reduce computational effort
- Secondary goal: lower detection latency

Approach by Schultz-Møller, Migliavacca & Pietzuch (DEBS 2009)

- Given high-level pattern description, select efficient automata-based plan
- Plan efficiency is quantified by cost model
- Cost model is based on arrival rates of event types
Generic Pattern Rewriting - Cost Model

Assumptions

- Asymptotic costs
- Predicate assumption is negligible (efficient indexing)
- Selectivity is ignored
- Event arrival follows Poisson process with known rate
- CPU cost is function of event rate only
Generic Pattern Rewriting - Cost Model

Assumptions

- Asymptotic costs
- Predicate assumption is negligible (efficient indexing)
- Selectivity is ignored
- Event arrival follows Poisson process with known rate
- CPU cost is function of event rate only

General idea

- Operator costs are additive, \( \text{cost}(Q) = \sum_{o \in Q} \text{cost}(o) \)
- Cost per operator is defined by event rates
Generic Pattern Rewriting - Cost Model cont.

The Union operator

- Union over two event types \( E1, E2 \)
- Cost \( \text{cost}(Q) = \mu(E1) + \mu(E2) \)
Generic Pattern Rewriting - Cost Model cont.

The Union operator

- Union over two event types $E_1, E_2$
- Cost $\text{cost}(Q) = \mu(E_1) + \mu(E_2)$

The Next operator

- Next over two event types $E_1, E_2$ with two predicates, $\varphi$ and $\theta$
- Cost considers bursting
  - Let $\mu(E_1) > \mu(E_2)$
  - Many partial instances terminated by single event
  - Burst size $b_s$ and rate $b_r$
- Cost $\text{cost}(Q) = \mu(E_1) + \frac{DT(E_2)}{DT(E_1)}b_s(E_1)b_s(E_2)b_r(E_2) + \max(b_r(E_2) - b_r(E_1), 0)b_s(E_2)$
Union rewriting

- Union is commutative and associative
- For Union with $n$ operators, there are $\frac{(2n)!}{n!}$ candidate plans
- Exploration of all of them infeasible for large $n$
- Greedy algorithm to select plan
Union rewriting

- Union is commutative and associative
- For Union with $n$ operators, there are $\frac{(2n)!}{n!}$ candidate plans
- Exploration of all of them infeasible for large $n$
- Greedy algorithm to select plan

```java
findOptimalUnion(e_pat)
    Q = extractUnionOperands(e_pat)
    n = Q.size()
    for(i = 0; i < n - 1; i++)
        l = Q.extractMinRate()
        r = Q.extractMinRate()
        newRate = l.rate() + r.rate()
        Q.insert(new UnionOperator(l, r), newRate)
    return Q.extractMinRate()
```
Example query: Union of four event types $E_1, E_2, E_3, E_4$
Next rewriting

- For Next with $n$ operators, there are $\frac{(2n)!}{n!}$ candidate plans
- Use dynamic programming
- Iteratively find optimal sub-expressions of increasing length
Next rewriting

- For Next with $n$ operators, there are $\frac{(2n)!}{n!}$ candidate plans
- Use dynamic programming
- Iteratively find optimal sub-expressions of increasing length

```java
findOptimalNext(e_pat)
    list = extractNextOperands(e_pat)
    n = list.size()
    for(l = 2; l <= n; l++)
        for(i = 0; i < n-l+1; i++)
            best = new InfinityCost()
            j = i+1
            for(k = i; k < j; k++)
                ele1 = list.get(index0f(i, k, n))
                ele2 = list.get(index0f(k+1, j, n))
                if(cost(ele1, ele2) < best.cost())
                    best = New NextOperator(ele1, ele2)
            list.add(best)
    return list.get(index0f(0, n-1, n))
```
Postponed Plan Evaluation - Overview

Setting

- Automata-based execution plans
- Non-determinism because of
  - Kleene Closure operator (+)
  - Skipping event processing policies
- Complexity of detecting single event is relatively low
- Complexity of detecting all events is comparably high

Example stream: a1, b1, b2, b3, c1
Creates 14 simultaneous runs / partial matches
Postponed Plan Evaluation - Overview

Setting

- Automata-based execution plans
- Non-determinism because of
  - Kleene Closure operator (+)
  - Skipping event processing policies
- Complexity of detecting single event is relatively low
- Complexity of detecting all events is comparably high
Postponed Plan Evaluation - Overview

Setting

- Automata-based execution plans
- Non-determinism because of
  - Kleene Closure operator (+)
  - Skipping event processing policies
- Complexity of detecting single event is relatively low
- Complexity of detecting all events is comparably high

Example stream: a1, b1, b2, b3, c1
Postponed Plan Evaluation - Overview

Setting
- Automata-based execution plans
- Non-determinism because of
  - Kleene Closure operator (+)
  - Skipping event processing policies
- Complexity of detecting single event is relatively low
- Complexity of detecting all events is comparably high

Example stream: a1, b1, b2, b3, c1
Creates 14 simultaneous runs / partial matches
Postponed Plan Evaluation - Approach

Approach by Zhang, Diao & Immerman (SIGMOD 2014)

- Based on plan evaluation operations:
  - Edge evaluation
  - Run initialization / extension / cloning / proceeding / termination
- Post-pone these operations if possible
- Avoid construction of runs in the matching phase
- Enumerate/unfold runs only in the result construction phase
Postponed Plan Evaluation - Approach

Approach by Zhang, Diao & Immerman (SIGMOD 2014)

- Based on plan evaluation operations:
  - Edge evaluation
  - Run initialization / extension / cloning / proceeding / termination
- Post-pone these operations if possible
- Avoid construction of runs in the matching phase
- Enumerate/unfold runs only in the result construction phase

Benefits

- Filtering of non-viable runs
- Fewer simultaneous runs reduce cost of process per event
- Sharing of cost of intermediate steps (e.g., edge evaluation)
Postponed Plan Evaluation - Example

Example stream: a1, b1, b2, b3, c1
Postponed Plan Evaluation - Example

Example stream:
\( a_1, b_1, b_2, b_3, c_1 \)
Postponed Plan Evaluation - Example

Example stream:
\texttt{a1, b1, b2, b3, c1}

<table>
<thead>
<tr>
<th>NFA^b</th>
<th>Postponing</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>(a1,-)</td>
</tr>
<tr>
<td>b1</td>
<td>((a1,b1,-)) ((a1,-))</td>
</tr>
<tr>
<td>b2</td>
<td>((a1,b1,b2,-)) ((a1,b1,-)) ((a1,b2,-))</td>
</tr>
<tr>
<td>b3</td>
<td>((a1,b1,b2,b3,-)) ((a1,b1,b2,-)) ((a1,b1,b3,-)) ((a1,b1,-)) ((a1,b2,b3,-)) ((a1,b2,-)) ((a1,b3,-)) ((a1,-))</td>
</tr>
<tr>
<td>c1</td>
<td>((a1,b1,b2,b3,c1)) ((a1,b2,b3,-)) ((a1,b1,b2,c1)) ((a1,b1,b2,-)) ((a1,b1,b3,c1)) ((a1,b1,b3,-)) ((a1,b1,c1)) ((a1,b1,-)) ((a1,b2,b3,c1)) ((a1,b2,b3,-)) ((a1,b2,c1)) ((a1,b2,-)) ((a1,b3,c1)) ((a1,b3,-)) ((a1,-))</td>
</tr>
</tbody>
</table>
Semantic Rewriting - Overview

Setting

- Event generation follows business process
- Business process describes activities that are executed in coordination
- Example domains: Business Activity Monitoring (BAM), Management of Service Level Agreements (SLAs)
Semantic Rewriting - Overview

Setting

▶ Event generation follows business process
▶ Business process describes activities that are executed in coordination
▶ Example domains: Business Activity Monitoring (BAM), Management of Service Level Agreements (SLAs)

Approach by Weidlich, Ziekow, Gal, Mendling & Weske (TKDE 2014)

▶ Extract behavioural constraints from process models
▶ Exploit that these constraints can be projected to event types
▶ Use this knowledge for rewriting of patterns and automata-based execution plans
Semantic Rewriting - Approach

Process Expert

- Process Models
  - Behavioral Profile Extraction
  - Process Engine

Business Analyst

- Monitoring Patterns
  - Alert if \( \text{all}(E_1, E_2) \)
- Pattern Translation
- Execution Plans
  - \( E_1 \rightarrow \text{pull} \ E_2 \)
Semantic Rewriting - Process Models

Process model

- Describes activities, executed in coordination, to reach a goal
- Logical and causal dependencies between activities
- Common control flow constructs are sequences, choices, concurrency, loops
- Instance of process model describes a specific case
Semantic Rewriting - Process Models

Process model

- Describes activities, executed in coordination, to reach a goal
- Logical and causal dependencies between activities
- Common control flow constructs are sequences, choices, concurrency, loops
- Instance of process model describes a specific case

Example of a process model given as a Petri-net
Semantic Rewriting - Constraint Model

Behavioural profile

- Set of binary relations over activities of process model
- Abstraction of behaviour, i.e., trace semantics
- $X < Y$, iff there is a trace in which $X$ occurs before $Y$
- Definition of relations
  - Exclusiveness, $X + Y$, iff $X \not< Y$ and $Y \not< X$
  - Order, $X \rightarrow Y$, iff $X < Y$ and $Y \not< X$
  - Interleaving, $X||Y$, iff $X < Y$ and $Y < X$
- Relations are mutually exclusive and partition Cartesian product of activities
- Relations are computed efficiently (structurally)
Semantic Rewriting - Constraint Model Example

- **A**: Get Contact
- **B**: Contact Customer
- **C**: Analyse Competitors
- **D**: Submit Quote
- **E**: Negotiate Contract
- **F**: Create Loss Report
- **G**: Close Deal
Semantic Rewriting - Constraint Model Example

A | B | C | D | E | F | G
---|---|---|---|---|---|---
A | + | → | → | → | → | → | →
B | ← | + | → | → | → | → | →
C | ← | + | → | → | → | → | →
D | ← | ← | + | → | → | → | →
E | ← | ← | ← | ← | + | ← | →
F | ← | ← | ← | ← | + | + | +
G | ← | ← | ← | ← | ← | + | +
Semantic Rewriting - Optimisation Procedure

Project relations to events of process instances
Semantic Rewriting - Optimisation Procedure

Project relations to events of process instances
Semantic Rewriting - Optimisation Procedure

Project relations to events of process instances

Actual optimisation is rule-based

Formal results on completeness, efficiency, and correctness of rules
**Sequentialisation Rule**

- CE refers to conjunction (all)
- All activities related to child events are ordered

Rewriting: \( E_1 \cup E_2 \Rightarrow SEQ(E_1, E_2) \)
Semantic Rewriting - Sequentialisation Rule

Sequentialisation Rule

- CE refers to conjunction (all)
- All activities related to child events are ordered

Rewriting: $E_1 \cup E_2 \Rightarrow SEQ(E_1, E_2)$

Example: $A \cup G \Rightarrow SEQ(A, G)$
Early Termination Rule

- CE is sequence of two events
- Insert third event, s.t. activity is ordered with first, exclusive to second

Rewriting: \( SEQ(E_1, E_2) \Rightarrow SEQ(E_1, \neg E_3, E_2) \)
Semantic Rewriting - Early Termination Rule

Early Termination Rule

- CE is sequence of two events
- Insert third event, s.t. activity is ordered with first, exclusive to second

Rewriting: $SEQ(E_1, E_2) \Rightarrow SEQ(E_1, \neg E_3, E_2)$

Example: $SEQ(A, G) \Rightarrow SEQ(A, \neg F, G)$
Optimisations Summary

The need for optimising event recognition

- Driven by throughput and latency requirements
- Computational effort and memory consumption
Optimisations Summary

The need for optimising event recognition
- Driven by throughput and latency requirements
- Computational effort and memory consumption

Various angles to optimise event recognition
- Pattern and plan rewriting
- Efficient plan execution
- Use of semantic information

Much more in the literature and actively researched
- Parallelisation of event processing
- Distributed event recognition (in-situ processing)
- Sampling of event streams
- Load shedding
- Heterogeneous hardware (GPGPUs, FPGAs)
Tutorial Structure

- Temporal reasoning systems.
- Optimisation.
- Open issues.
Issue (1): Multi-scale Temporal Aggregation of Events

Composite events evolve over multiple scales of time and space

- Actual low-level events
- Context information, e.g., historic data
Issue (1): Multi-scale Temporal Aggregation of Events

Composite events evolve over multiple scales of time and space

- Actual low-level events
- Context information, e.g., historic data

Recognition system should be adaptable, computing dynamically the appropriate scales

- Appropriate lengths of multi-granular windows
- Appropriate temporal event density
- Appropriate slice of context information
Issue (1): Multi-scale Temporal Aggregation of Events

Composite events evolve over multiple scales of time and space
  ▶ Actual low-level events
  ▶ Context information, e.g., historic data

Recognition system should be adaptable, computing dynamically the appropriate scales
  ▶ Appropriate lengths of multi-granular windows
  ▶ Appropriate temporal event density
  ▶ Appropriate slice of context information

Semantics and processing guarantees, despite adaptation
Issue (2): Real-time Probabilistic Event Recognition

Types of uncertainty

- Event occurrence is uncertain
- Complex event definition is uncertain

Existing solution

- Probabilistic graphical models & logic-based approaches

Major drawback of existing solutions

- Large overhead that does not allow for real-time performance
Issue (3): Distributed Event Recognition

Distribution of event recognition among multiple nodes
- Distribution of detection automata
- Distribution of run-time computation
- Exploiting semantic dependencies between event queries
- Query rewriting to obtain independent sub-queries

But, virtually no work on distributing uncertain event recognition
Interplay with communication efficiency
- Reduce communication by decomposing recognition in set of local constraints at event sources
  - By sketching
  - By geometric models
- Limited to particular types of composite events: functions over aggregate values
Issue (4): Getting Domain Experts in the Loop

Event recognition needs to be defined at higher levels of abstraction. Domain experts should be able to define complex events for analytics.
Issue (4): Getting Domain Experts in the Loop

Event recognition needs to be defined at higher levels of abstraction.
Issue (4): Getting Domain Experts in the Loop

Event recognition needs to be defined at higher levels of abstraction.

Domain experts should be able to define complex events for analytics.
Issue (4): Getting Domain Experts in the Loop

Event recognition needs to be defined at higher levels of abstraction.

Domain experts should be able to define complex events for analytics.
Issue (4): Getting Domain Experts in the Loop

Event recognition needs to be defined at higher levels of abstraction.

Domain experts should be able to define complex events for analytics.
Issue (5): Event Forecasting and Prediction

Current work focuses on

- Past (post mortem analysis)
- Present (monitoring)
Issue (5): Event Forecasting and Prediction

Current work focuses on

- Past (post mortem analysis)
- Present (monitoring)

Stream processing will focus on future

- Predictive analysis
- Trend detection
- Situation-awareness
Issue (5): Event Forecasting and Prediction

Current work focuses on

▶ Past (post mortem analysis)
▶ Present (monitoring)

Stream processing will focus on future

▶ Predictive analysis
▶ Trend detection
▶ Situation-awareness

The earlier the better

▶ Time to react to an anticipated situation
▶ Avoidance of situation or initialisation of counter measures