Tutorial:
Complex Event Recognition Languages

Alexander Artikis\textsuperscript{1,2}, Alessandro Margara\textsuperscript{3}, Martin Ugarte\textsuperscript{4}, Stijn Vansummeren\textsuperscript{4}, Matthias Weidlich\textsuperscript{5}

\textsuperscript{1}University of Piraeus, \textsuperscript{2}NCSR Demokritos, \textsuperscript{3}Politecnico di Milano, \textsuperscript{4}Université Libre de Bruxelles, \textsuperscript{5}Humboldt-Universität zu Berlin
Complex Event Recognition (Event Pattern Matching)

**Input**: Streams of SDEs

**Recognition**: Event Recognition System

**Output**: Recognised CEs

**CE Definitions**
Complex Event Recognition for Security
# Online Recognition

<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 coord(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 coord(id$_2$) = (25.88, -19.80)</td>
<td></td>
</tr>
<tr>
<td>340 active(id$_1$)</td>
<td></td>
</tr>
<tr>
<td>340 coord(id$_1$) = (20.88, -11.90)</td>
<td></td>
</tr>
<tr>
<td>340 walking(id$_3$)</td>
<td></td>
</tr>
<tr>
<td>340 coord(id$_3$) = (24.78, -18.77)</td>
<td></td>
</tr>
<tr>
<td>380 walking(id$_3$)</td>
<td></td>
</tr>
<tr>
<td>380 coord(id$_3$) = (27.88, -9.90)</td>
<td></td>
</tr>
<tr>
<td>380 walking(id$_2$)</td>
<td></td>
</tr>
<tr>
<td>380 coord(id$_2$) = (28.27, -9.66)</td>
<td></td>
</tr>
</tbody>
</table>
### Online Recognition

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>340 inactive(id&lt;sub&gt;0&lt;/sub&gt;)</td>
<td>since(340) leaving object(id&lt;sub&gt;1&lt;/sub&gt;, id&lt;sub&gt;0&lt;/sub&gt;)</td>
</tr>
<tr>
<td>340 coord(id&lt;sub&gt;0&lt;/sub&gt;) = (20.88, -11.90)</td>
<td></td>
</tr>
<tr>
<td>340 appear(id&lt;sub&gt;0&lt;/sub&gt;)</td>
<td></td>
</tr>
<tr>
<td>340 walking(id&lt;sub&gt;2&lt;/sub&gt;)</td>
<td></td>
</tr>
<tr>
<td>340 coord(id&lt;sub&gt;2&lt;/sub&gt;) = (25.88, -19.80)</td>
<td></td>
</tr>
<tr>
<td>340 active(id&lt;sub&gt;1&lt;/sub&gt;)</td>
<td></td>
</tr>
<tr>
<td>340 coord(id&lt;sub&gt;1&lt;/sub&gt;) = (20.88, -11.90)</td>
<td></td>
</tr>
<tr>
<td>340 walking(id&lt;sub&gt;3&lt;/sub&gt;)</td>
<td></td>
</tr>
<tr>
<td>340 coord(id&lt;sub&gt;3&lt;/sub&gt;) = (24.78, -18.77)</td>
<td></td>
</tr>
<tr>
<td>380 walking(id&lt;sub&gt;3&lt;/sub&gt;)</td>
<td></td>
</tr>
<tr>
<td>380 coord(id&lt;sub&gt;3&lt;/sub&gt;) = (27.88, -9.90)</td>
<td></td>
</tr>
<tr>
<td>380 walking(id&lt;sub&gt;2&lt;/sub&gt;)</td>
<td></td>
</tr>
<tr>
<td>380 coord(id&lt;sub&gt;2&lt;/sub&gt;) = (28.27, -9.66)</td>
<td></td>
</tr>
</tbody>
</table>
### Online Recognition

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>340 inactive(id₀)</td>
<td>since(340) leaving_object(id₁, id₀)</td>
</tr>
<tr>
<td>340 coord(id₀)</td>
<td>since(340) moving(id₂, id₃)</td>
</tr>
<tr>
<td>340 appear(id₀)</td>
<td></td>
</tr>
<tr>
<td>340 walking(id₂)</td>
<td></td>
</tr>
<tr>
<td>340 coord(id₂)</td>
<td></td>
</tr>
<tr>
<td>340 active(id₁)</td>
<td></td>
</tr>
<tr>
<td>340 coord(id₁)</td>
<td></td>
</tr>
<tr>
<td>340 walking(id₃)</td>
<td></td>
</tr>
<tr>
<td>340 coord(id₃)</td>
<td></td>
</tr>
<tr>
<td>380 walking(id₃)</td>
<td></td>
</tr>
<tr>
<td>380 coord(id₃)</td>
<td></td>
</tr>
<tr>
<td>380 walking(id₂)</td>
<td></td>
</tr>
<tr>
<td>380 coord(id₂)</td>
<td></td>
</tr>
</tbody>
</table>
Application Requirements

- Input:
  - Instantaneous events.
  - Context information.
  - The interval may be open.
  - Relational events.
  - No limit on the temporal distance between the events comprising the composite activity (no 'WITHIN' constraint).
  - Concurrency constraints.
  - Spatial reasoning.
  - Event hierarchies.
Application Requirements

- **Input:**
  - Instantaneous events.
  - Context information.

- **Output:** durative events.
  - The interval may be open.
Application Requirements

- **Input:**
  - Instantaneous events.
  - Context information.

- **Output:** durative events.
  - The interval may be open.
  - Relational events.

- No limit on the temporal distance between the events comprising the composite activity (no 'WITHIN' constraint).
- Concurrency constraints.
- Spatial reasoning.
- Event hierarchies.
Application Requirements

- **Input:**
  - Instantaneous events.
  - Context information.

- **Output:** durative events.
  - The interval may be open.
  - Relational events.
  - No limit on the temporal distance between the events comprising the composite activity (no ‘WITHIN’ constraint).
Application Requirements

- **Input:**
  - Instantaneous events.
  - Context information.

- **Output:** durative events.
  - The interval may be open.
  - Relational events.
  - No limit on the temporal distance between the events comprising the composite activity (no ‘WITHIN’ constraint).
  - Concurrency constraints.
Application Requirements

- **Input:**
  - Instantaneous events.
  - Context information.

- **Output:** durative events.
  - The interval may be open.
  - Relational events.
  - No limit on the temporal distance between the events comprising the composite activity (no ‘WITHIN’ constraint).
  - Concurrency constraints.
  - Spatial reasoning.
Application Requirements

- **Input:**
  - Instantaneous events.
  - Context information.

- **Output:** durative events.
  - The interval may be open.
  - Relational events.
  - No limit on the temporal distance between the events comprising the composite activity (no ‘WITHIN’ constraint).
  - Concurrency constraints.
  - Spatial reasoning.
  - Event hierarchies.
Complex Event Recognition for Maritime Surveillance
Complex Event Recognition for Maritime Surveillance
Fast Approach

- A vessel is moving at a high speed ...
- towards other vessels.
Suspicious Delay

- A vessel fails to report position ...
- and the estimated speed during the communication gap is low.
Possible Rendezvous

- Two vessels are suspiciously delayed ...
- in the same location ...
- at the same time.
Application Requirements

- Input:
  - Instantaneous events.
  - Durative events.
  - Context information.
Application Requirements

- **Input:**
  - Instantaneous events.
  - Durative events.
  - Context information.

- **Output:** Durative events.
  - The interval may be open.
  - Relational events.
  - No limit on the temporal distance between the events comprising the composite activity.
  - Concurrency constraints.
  - Spatial reasoning.
  - Event hierarchies.
Complex Event Recognition for Credit Card Fraud Management

Input:

- Credit card transactions from all over the world.

Output:

- Cloned card — a credit card is being used simultaneously in different countries.
- New high use — the card is being frequently used in merchants or countries never used before.
- Potential batch fraud — many transactions from multiple cards in the same point-of-sale terminal in high amounts.
Application Requirements

- **Input:**
  - Instantaneous events.
  - Context information.

- **Output:** durative events.
  - Relational & non-relational events.
  - Limited temporal distance between the events comprising fraudulent activity (‘WITHIN’ constraint).
  - Event sequences.
  - Spatial reasoning for some patterns.
Agricultural Monitoring
Application Requirements

- **Input**: Instantaneous events.
- **Output**: Instantaneous/durative events.
  - Correlation of events of different sensors.
  - Limited temporal distance between the events comprising alarming behavior (‘WITHIN’ constraint).
  - Event sequences ('increasing streak of temperature measurements')
  - Spatial reasoning for some patterns.
Tutorial Outline

1. Automata-based models and methods
2. Tree-based models and methods
3. Logic-based models and methods
4. Outlook
Automata-based Models and Methods
Plantations might be at risk if sensor 1 detects low temperatures after a high-humidity period.

A wood fire is likely if low humidity is measured after high temperatures.

Similar to regular expressions

Use automata to detect complex events.
Challenges

**Regex th**

- A symbol $t$ immediately followed by a symbol $h$

**A wood fire is likely if low humidity is measured after high temperatures**

- An event $T$ eventually followed by an event $H$
A wood fire is likely if low humidity is measured after high temperatures.
Plantations might be at risk if sensor 1 detects low temperatures after a high-humidity period followed by a low-temperature event $T$.

High-humidity events $H_1, H_2, \ldots, H_n$.
Plantations might be at risk if sensor 1 detects low temperatures after a high-humidity period.

<table>
<thead>
<tr>
<th>Iterations (Kleene closure)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[1], {2}$</td>
</tr>
<tr>
<td>$[1], {3}$</td>
</tr>
<tr>
<td>$[1], {6}$</td>
</tr>
<tr>
<td>$[1], {7}$</td>
</tr>
<tr>
<td>$[1, 4], {6}$</td>
</tr>
<tr>
<td>$[1, 4], {7}$</td>
</tr>
<tr>
<td>$[1, 5], {6}$</td>
</tr>
<tr>
<td>$[1, 5], {7}$</td>
</tr>
<tr>
<td>$[1, 4, 5], {6}$</td>
</tr>
<tr>
<td>$[1, 4, 5], {7}$</td>
</tr>
<tr>
<td>$[4], {6}$</td>
</tr>
<tr>
<td>$[4], {7}$</td>
</tr>
<tr>
<td>$[4, 5], {6}$</td>
</tr>
<tr>
<td>$[4, 5], {7}$</td>
</tr>
</tbody>
</table>

$httthhttt$

$\begin{array}{cccccccc}
80 & 15 & 14 & 85 & 84 & 10 & 12 \\
1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}$
CAYUGA

*Towards Expressive Publish/Subscribe Systems.* Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald and Walker White; EDBT 2006

Cayuga Event Language (CEL)

Named streams

Attributes

\{ 
SELECT \textit{<attributes>} 
FROM \textit{<stream expression>} 
PUBLISH \textit{<output_stream_name>} 
\}

\textit{generates a stream}

\textit{has}

New stream named \textit{<output_stream_name>}
Projection, renaming

```
SELECT id as t_id
FROM Temp
PUBLISH TempIds
```
Filtering

Remove all tuples with temperatures below 30 degrees

\[
\text{FILTER}\{\text{temp} \geq 30\} \ \text{Temp}
\]

Stream expression

\[
\begin{align*}
\text{SELECT} & \quad \text{id, temp} \\
\text{FROM} & \quad \text{FILTER}\{\text{temp} \geq 30\}\ \text{Temp} \\
\text{PUBLISH} & \quad \text{HighTemp}
\end{align*}
\]

\[
\text{FILTER}\{<\text{condition}>\}<\text{stream_expression}>
\]

Stream expression
SELECT id, temp
FROM FILTER {temp >= 30} Temp
PUBLISH HighTemp
Select the (next) temperature of areas in which a low humidity was detected

Output schema: (id_1, id_2, hum, temp)
Hum $\{H(id=2,\ hum=15)\}_1$
$\{H(id=1,\ hum=12)\}_3$
$\{H(id=3,\ hum=22)\}_5$
$\{H(id=2,\ hum=14)\}_7$
$\{H(id=1,\ hum=18)\}_9$
...

Temp $\{T(id=1,\ temp=24)\}_2$
$\{T(id=2,\ temp=20)\}_4$
$\{T(id=3,\ temp=22)\}_6$
$\{T(id=1,\ temp=25)\}_8$
$\{T(id=2,\ temp=32)\}_{10}$
...

Hum $\text{NEXT}$\{$1.id = 2.id \ AND \ 1.hum < 25$\} Temp

(id_1=2, id_2=2, hum=15, temp=20)
(id_1=1, id_2=1, hum=12, temp=25)
(id_1=3, id_2=3, hum=22, temp=22)
(id_1=2, id_2=2, hum=14, temp=25)
...

Compile into an automaton?
Hum \{ H(id=2, hum=15) \_1 \\
H(id=1, hum=12) \_3 \\
H(id=3, hum=22) \_5 \\
H(id=2, hum=14) \_7 \\
H(id=1, hum=18) \_9 \\ ...
\}

Temp \{ T(id=1, temp=24) \_2 \\
T(id=2, temp=20) \_4 \\
T(id=3, temp=22) \_6 \\
T(id=1, temp=25) \_8 \\
T(id=2, temp=32) \_10 \\ ...
\}

Hum \text{ NEXT}\{ $1.id = $2.id \text{ AND } $1.hum < 25 \} \text{ Temp} 

\text{Temp } t, \text{ Hum } h \\
t.id \neq x

\text{Hum } h \\
h.hum < 25 \\
x = h.id

\text{Temp } t, \text{ Hum } h \\
t.id = x

start \rightarrow q\_1 \xrightarrow{\text{output } h.*} q\_2 \xrightarrow{\text{output } t.*} q\_3
Hum

\{ H(id=2, hum=15) \_1
H(id=1, hum=12) \_3
H(id=3, hum=22) \_5
H(id=2, hum=14) \_7
H(id=1, hum=18) \_9
\ldots
\}

Temp

\{ T(id=1, temp=24) \_2
T(id=2, temp=20) \_4
T(id=3, temp=22) \_6
T(id=1, temp=25) \_8
T(id=2, temp=32) \_10
\ldots
\}

$q_1$ \quad $H_1$ \quad $q_2$ \quad $q_2$ \quad $T_4$ \quad $q_3$

\{H$_1$, T$_4$\}

Temp $t$, Hum $h$
\[ t.\text{id} \neq x \]

$q_1$ \quad \text{start} \quad $q_2$ \quad \text{output h.$*$} \quad $q_2$ \quad $q_3$ \quad \text{output t.$*$}

Hum $h$
\[ h.\text{hum} < 25 \]
\[ x = h.\text{id} \]

Temp $t$
\[ t.\text{id} = x \]
Temp

T(id=1, temp=24)
T(id=2, temp=20)
T(id=3, temp=22)
T(id=1, temp=25)
...

Hum

H(id=2, hum=15)
H(id=1, hum=12)
H(id=3, hum=22)
H(id=2, hum=14)
H(id=1, hum=18)
...

\begin{array}{ll}
\text{Hum} & \{H(id=2, hum=15), H(id=1, hum=12), H(id=3, hum=22), H(id=2, hum=14), H(id=1, hum=18), \ldots\} \\
\text{Temp} & \{T(id=1, temp=24), T(id=2, temp=20), T(id=3, temp=22), T(id=1, temp=25), T(id=2, temp=32), \ldots\} \\
\end{array}

\begin{equation}
\begin{aligned}
&\text{Hum } h \\
&h.hum < 25 \\
&x = h.id \\
&\text{output } h.* \\
\end{aligned}
\end{equation}

\begin{equation}
\begin{aligned}
&\text{Temp } t, \text{ Hum } h \\
&t.id \neq x \\
&\text{output } t.* \\
\end{aligned}
\end{equation}

\begin{array}{ll}
\text{q1} & \text{q1} \\
\text{q1} & \text{q2} \\
\text{H3} & \text{x = 15} \\
\text{q2} & \text{q2} \\
\text{q2} & \text{T8} \\
\text{q3} & \\
\end{array}

\begin{array}{ll}
\text{start} & \text{q1} \\
& \text{output } h.* \\
\text{Hum } h \\
&h.hum < 25 \\
&x = h.id \\
\text{q2} & \text{q2} \\
& \text{output } t.* \\
\text{T8} & \text{q3} \\
& \text{Temp } t \\
&t.id = x \\
\end{array}

\{H3, T8\}
Select every element from <Stream $1> and the next element from <Stream $2> satisfying the <condition>.
Iteration

<Stream $1>
NEXT{<condition>}  <Stream $2>
...
NEXT{<condition>}  <Stream $2>

<condition> for filtering <Stream $2> (like in NEXT)?

Condition to stop iterations?

Computations during iterations…
Iteration

<Stream $1> FOLD{<NEXT_condition>, <stop_condition>, <compute>} <Stream $2>

Conditions have access to $1$, $2$, and $. The latter refers to the *previous* iteration.

<compute> can do incremental computations
Iteration

Create a stream of all decreasing temperature streaks composed of more than three events.

FILTER{cnt > 3}(
  (SELECT *, 1 AS cnt FROM Temp)
  FOLD{$.temp > $2.temp, 
  $.temp <= $2.temp, 
  $.cnt + 1 AS cnt } Temp
)

FILTER\{cnt > 3\}( 
(SELECT *, 1 AS cnt FROM Temp)
FOLD\{$.temp > $2.temp, 
    $.temp <= $2.temp, 
    $.cnt + 1 AS cnt \} Temp 
)
(id_1=1, temp_1=24, id_2=1, temp_2=17, cnt=4)

(id_1=2, temp_1=20, id_2=1, temp_2=17, cnt=3)

X

(id_1=2, temp_1=20, id_2=3, temp_2=17, cnt=2)

FILTER\{cnt > 3\}( (SELECT *, 1 AS cnt FROM Temp) FOLD\{$.temp > $2.temp, 
$.temp <= $2.temp, 
$.cnt + 1 AS cnt \} Temp )
Compilation

Notify me of decreasing temperature streaks of at least 3 events in which the temperature drops from above 40 to below 20.
FILTER\{cnt \geq 3, temp < 20\}( (SELECT *, 1 AS cnt FROM FILTER\{temp > 40\}Temp) 

FOLD\{$.temp > $2.temp, 
$.temp \leq $2.temp \text{ OR } 
(cnt \geq 3 \text{ AND } $.temp < 20), 
$.cnt + 1 \text{ AS } cnt\} 

) Temp
Temp \( t \)
\[ t.\text{temp} > 40 \]
\[ x = t.\text{temp} \]
\[ \text{cnt} = 1 \]

Temp \( t \)
\[ t.\text{temp} < x \]
\[ x = t.\text{temp} \]
\[ \text{cnt} += 1 \]

Temp \( t \)
\[ t.\text{temp} \geq x \]
\[ \text{cnt} \geq 3 \]
\[ x < 20 \]

\[ x = 45 \]
\[ \text{cnt} = 1 \]

\[ x = 35 \]
\[ \text{cnt} = 2 \]
\( q_1 \) \( q_2 \) \( q_3 \)

**Temp t**
- \( t.\text{temp} > 40 \)
- \( x = t.\text{temp} \)
- \( \text{cnt} = 1 \)

**Hum h**
- \( t.\text{temp} \geq x \)
- \( \text{cnt} \geq 3 \)
- \( x < 20 \)

**Output**
- \( t.*, \text{cnt} \)

**Input sequence**
- \( t t h t t h t h t h t t t t \)

- \( 45 \ 35 \ 40 \ 36 \ 20 \ 43 \ 31 \ 19 \ 42 \ 55 \ 29 \ 18 \ 23 \)
A simple (?) example

Plantations might be at risk if sensor 1 detects low temperatures after a high-humidity period.

$\text{FILTER}\{\text{temp} < 10\}(\text{SELECT} *, 1 \text{ AS cnt FROM} \text{Hum})$

$\text{FOLD}\{\text{cnt} + 1 \text{ AS cnt}\}$

(\text{Hum NEXT}\{\text{True}\}\text{ Temp})

Will never be selected!
A simple (?) example

Plantations might be at risk if sensor 1 detects low temperatures after a high-humidity period.

Writing rules is not a simple task…

Will never be selected!
SEMANTICS
SASE / SASE+

On Complexity and Optimization of Expensive Queries in Complex Event Processing. Haopeng Zhang, Yanlei Diao, and Neil Immerman; SIGMOD 2014


High-Performance Complex Event Processing over Streams. Eugene Wu, Yanlei Diao and Shariq Rizvi; SIGMOD 2006
Language

Single stream of timestamped events

Named relations with attributes

\[
\text{EVENT} \ <\text{event\_pattern}> \\
[\text{WHERE} \ <\text{filter}>] \\
[\text{WITHIN} \ <\text{time\_window}>]
\]

New stream, events contain all attributes (no projection)
Sequencing

\[ T_1(\text{id}=1, \text{temp}=24, \text{tstamp}=0.5) \]
\[ H_1(\text{id}=2, \text{hum}=25, \text{tstamp}=0.65) \]
\[ H_2(\text{id}=1, \text{hum}=33, \text{tstamp}=1) \]
\[ T_2(\text{id}=2, \text{temp}=25, \text{tstamp}=1.32) \]
\[ H_3(\text{id}=2, \text{hum}=21, \text{tstamp}=1.34) \]
\[ \ldots \]

**EVENT** \( \text{SEQ}(T \ t, H \ h) \)

“All pairs (temperature, humidity), such that temperature occurred before humidity”

\[ \{T_1, H_1\}, \{T_1, H_2\}, \{T_1, H_3\}, \{T_2, H_3\} \]
\{T_1, H_1\}, \{T_1, H_2\}, \{T_1, H_3\}, \{T_2, H_3\}

T_1(id=1, temp=24, tstamp=0.5)
H_1(id=2, hum=25, tstamp=0.65)

TH_1(id_1=1, id_2=2, temp=24, hum=25, tstamp=0.65)
Filtering

\[ T_1(\text{id}=1, \text{temp}=24, \text{tstamp}=0.5) \]
\[ H_1(\text{id}=2, \text{hum}=25, \text{tstamp}=0.65) \]
\[ H_2(\text{id}=1, \text{hum}=33, \text{tstamp}=1) \]
\[ T_2(\text{id}=2, \text{temp}=25, \text{tstamp}=1.32) \]
\[ H_3(\text{id}=2, \text{hum}=21, \text{tstamp}=1.34) \]

**EVENT** \( \text{SEQ}(T \ t; H \ h) \)

**WHERE** \( t.\text{id} = h.\text{id} \)

“All pairs (\text{temperature, humidity}), with the same \text{id} such that \text{temperature} occurred before \text{humidity}”

\{\text{T}_1, \text{H}_2\}, \{\text{T}_2, \text{H}_3\}
Time Windows

\[ T_1(id=1, \text{temp}=24, \text{tstamp}=0.5) \]
\[ H_1(id=2, \text{hum}=25, \text{tstamp}=0.65) \]
\[ H_2(id=1, \text{hum}=33, \text{tstamp}=1) \]
\[ T_2(id=2, \text{temp}=25, \text{tstamp}=1.32) \]
\[ H_3(id=2, \text{hum}=45, \text{tstamp}=1.34) \]

**EVENT** \( \text{SEQ}(T \ t, H \ h) \)

**WHERE** \( t.id = h.id \) \{\( T_2 \), \( H_3 \)\}

**WITHIN** 0.1 seconds

“All pairs (temperature, humidity), with the same \text{id} such that temperature occurred before humidity, but at most 0.1 seconds before”
Negation

\[ T_1(\text{id}=1, \text{temp}=24, \text{tstamp}=0.5) \]
\[ H_1(\text{id}=2, \text{hum}=25, \text{tstamp}=0.65) \]
\[ H_2(\text{id}=1, \text{hum}=33, \text{tstamp}=1) \]
\[ T_2(\text{id}=2, \text{temp}=25, \text{tstamp}=1.32) \]
\[ H_3(\text{id}=2, \text{hum}=45, \text{tstamp}=1.34) \]

\{T_2, H_3\}

**EVENT**  \( \text{SEQ}(T \ t, \ !H \ h1, \ H \ h2) \)

**WHERE**  \( t.\text{id} = h2.\text{id1} \)

“All pairs (temperature, humidity), with the same id such that humidity is the first humidity measurement after temperature”
SASE+

Multiple streams

Kleene closure

FROM <input_stream>
[PATTERN <event_pattern>]
[WHERE <filter>]
[WITHIN <time_window>]
[HAVING <pattern_condition>]
OUTPUT <output_stream_name>
Plantations might be at risk if sensor 1 detects low temperatures after a high-humidity period.

```
FROM HUM_TEMP_STREAM
PATTERN SEQ(H+ h[], T t)
WHERE h.id = 1 AND t.id = 1 AND
    h.hum > 70 AND t.temp < 10
WITHIN 5 minutes
HAVING count(h) >= 3
OUTPUT Plantation Risk
```
Plantations might be at risk if sensor 1 detects low temperatures after a high-humidity period.

```
FROM HUM_TEMP_STREAM
PATTERN SEQ (H+h[], T+t)
WHERE h.id = 1 AND t.id = 1 AND h.hum > 70 AND t.temp < 10
WITHIN 5 minutes
HAVING count(h) >= 3
OUTPUT Plantation Risk
```

More declarative
FROM HUM_TEMP_STREAM

PATTERN SEQ(H+ h[], T t)
WHERE h.id = 1 AND t.id = 1 AND
    h.hum > 70 AND t.temp < 10
WITHIN 5 minutes
HAVING count(h) >= 3
OUTPUT Plantation Risk
FROM HUM_TEMP_STREAM
PATTERN SEQ(H+ h[], T t)
WHERE h.id = 1 AND t.id = 1 AND
     h.hum > 70 AND t.temp < 10
     WITHIN 5 minutes
HAVING count(h) >= 3
OUTPUT Plantation Risk
What about negation?

```
FROM HUM_TEMP_STREAM
PATTERN SEQ(H+ h[], !T t1, T t2)
WHERE h.id = 1 AND t1.id = 1 AND t2.id = 1 AND h.hum > 70 AND t1.temp >= 10 AND t2.temp < 10
WITHIN 5 minutes
HAVING count(h) >= 3
OUTPUT Plantation Risk
```
SEMANTICS
Nondeterminism?

Graph:

- **q1**
  - Action: Output
  - Condition:
    - Hum h
    - h.hum > 70
    - x = h.time
    - cnt = 1
    - sum_h = h.hum

- **q2**
  - Action: Output
  - Condition:
    - Hum h
    - h.hum > 70
    - cnt += 1
    - sum_h += h.hum

- **q3**
  - Action: Output
  - Condition:
    - Temp t
    - t.temp < 10
    - t.time - x <= 5 min
    - cnt >= 3
Nondeterminism?

We need to run this!
### Event Processing

**Event a**
- **Condition**: $a.val > 20$
- **Action**: $x = a.val$

**Event b**
- **Condition**: $b.val > 25$

**Event c**
- **Condition**: $c.val > x$

---

**Transition Diagram**

- **Start State**: $q_1$
- **First Transition**: $a$ with condition $a.val > 20$
  - **Action**: $x = a.val$
  - **Next State**: $q_2$
- **Second Transition**: $b$ with condition $b.val > 25$
- **Third Transition**: $c$ with condition $c.val > x$
  - **Output**: $x$

---

**Order of Events**

- $a$, $b$, $c$, $c$, $a$, $b$, $c$, $c$

---

**Input Sequence**

- $25$, $35$, $78$, $10$, $20$, $30$, $76$, $27$, $23$, $6$, $68$, $28$
\( a \) \( b \) \( c \) \( a \) \( b \) \( a \) \( c \) \( b \) \( a \) \( b \) \( c \) \( c \)

\[
\begin{array}{c}
1 \\
25 \\
35 \\
78 \\
10 \\
20 \\
30 \\
76 \\
27 \\
23 \\
6 \\
68 \\
28 \\
\end{array}
\]

\[
\begin{array}{c}
1 \quad x = 25 \\
2 \quad 6 \quad x = 30 \\
3 \quad 9 \quad x = 23 \\
\end{array}
\]

\[
\begin{array}{c}
1 \\
2 \\
3 \\
\end{array}
\]

\[
\begin{array}{c}
2 \\
8 \\
7 \quad 76 \\
11 \quad 68 \\
12 \quad 28 \\
\end{array}
\]

\[
\begin{array}{c}
1 \quad [8] \\
1 \quad [8, 2] \\
1 \quad [2] \\
\end{array}
\]

\[
\begin{array}{c}
\text{event } a \\
an.val > 20 \\
x = a.val \\
\end{array}
\]

\[
\begin{array}{c}
\text{event } c \\
c.val > x \\
\text{output} \\
\end{array}
\]

\[
\begin{array}{c}
\text{event } b \\
b.val > 25 \\
\text{output} \\
\end{array}
\]
Correlation has to be considered during the output processing.

1. Start
   - Event a: a.val > 20
   - Output: x = a.val

2. Event b: b.val > 25
   - Output

3. Event c: c.val > x
   - Output

Output sequences:
- \(\{1, [8], 12\}\)
- \(\{1, [8, 2], 12\}\)

Correlation has to be considered during the output processing.
Conclusions

CER operators: sequencing, iteration, filtering, …

Multiple effective syntaxes proposed

Semantics (and sometimes syntax) not always clear

Previous research: deal with nondeterminism at runtime
Unified Automata Model

- Transitions evaluate formulas: **Symbolic Automata**
- Need to store variables: **Register Automata**
- Need to produce output: **Transducer**
Open Questions

Is there a general evaluation strategy for this model?

What fragments of automata can be run efficiently?

Is there a language capturing the computational model?

What is the complexity of compiling queries into automata?

How do different operators affect this complexity?
Part II
Recap of the First Part

Operators for CER
- Sequencing
- Kleene Closure
- Negation
- Filtering (predicates, time windows)

Semantics of queries given by automata

PATTERN SEQ(H+ h[], T t)
WHERE h.id = 1 AND t.id = 1
AND h.hum > 70 AND t.temp < 10
WITHIN 5 minutes
HAVING count(h) >= 3
1. Introduction
2. Common CER operators and automata-based CER models
3. Tree-based models and methods
4. Logic-based models and methods
5. Outlook
Tree-based Models Overview

Tree-based models may be used
- At design-time:
  Event pattern specification
- At run-time:
  Query plan for actual event recognition

Tree-based models are used in various systems
- Initial version of SASE
- ZStream
- Esper


N.P. Schultz-Møller, M. Migliavacca, P.R. Pietzuch: *Distributed complex event processing with query rewriting*. DEBS 2009
Chaining of Pattern Operators

Query plans as a fixed operator tree

- Combine automata-based models with additional operators
- Example: Initial version of SASE (SIGMOD 2006):

EVENT SEQ(A x₁,B x₂,!C x₃,D x₄)
WHERE [a₁, a₂] AND x₁.a₃ = 1
AND x₁.a₄ < x₄.a₄
WITHIN T
Fixed recognition order

- Order of events in the definition of a pattern (pattern order) is also followed in the recognition procedure (recognition order)
- Problematic from a performance point of view if there are large differences in selectivities

EVENT SEQ((H+ h[], P+ p[], T t))
Forced sequential order

- Automata enforces ordering of events
- Pattern may not define any ordering

**EVENT UNION** (H h, T t)
Integration of negation

- Terminal states are applicable solely if correlation predicates can be evaluated with past events
- Consequence: Either limited expressiveness or need for additional filtering

FROM HUM_TEMP_STREAM
PATTERN SEQ(H+ h[], !T t1, T t2)
WHERE h.id = 1 AND t1.id = 1 AND
t2.id = 1 AND h.hum > 70 AND
t1.temp < t2.temp + 5 AND t2.temp < 10
WITHIN 5 minutes
HAVING count(h) >= 3
OUTPUT Plantation Risk
How to use tree-based models for event pattern specification?
Tree-based Event Patterns

Main idea: Event pattern = operator tree

- **Leaf nodes**: types of individual events to be recognised
- **Non-leaf nodes**: composite type, built from the types of child nodes
- **Common composite types**:
  - Sequence, conjunction, and disjunction
  - Negation, in combination with the above
  - Kleene closure as a trinary operator (start, closure, end)

Different representations of a single pattern

- Nesting of operators
- Predicates at different levels of the tree
Example Pattern

A wood fire is likely if low humidity is measured after high temperatures

EVENT SEQ(T t, H h)
WHERE t.id = h.id
WITHIN 0.1

"All pairs (temperature, humidity), with the same id such that temperature occurred before humidity, but at most 0.1 seconds before"
Extended Example Pattern

\[
\text{EVENT SEQ}(T \ t, \ H \ h1, \ H \ h2) \\
\text{WHERE} \ t.id = h1.id \\
\quad \text{AND} \ h1.id = h2.id \\
\quad \text{AND} \ h1.temp > 40 \\
\quad \text{AND} \ h2.hum \leq 25 \\
\quad \text{AND} \ h3.hum \leq 25 \\
\quad \text{WITHIN} \ 0.5 \\
\]

“All triples (temperature, humidity, humidity) with the same id such that high temperature is followed by multiple low humidity within 0.5 seconds”
Extended Example Pattern Again

Left-deep pattern

T1: T
T2: H
T3: H

T1.id = T2.id
T2.id = T3.id
T3.ts - T1.ts ≤ 0.5sec
T1.temp > 40
T2.hum ≤ 25
T3.hum ≤ 25

Right-deep pattern

T1: T
T2: H
T3: H

T1.id = T2.id
T2.id = T3.id
T3.ts - T1.ts ≤ 0.5sec
T2.hum ≤ 25
T3.hum ≤ 25
T1.temp > 40
Example Pattern with Negation

EVENT SEQ(T t, !H h1, H h2)
WHERE t.id = h2.id
WITHIN 0.5

“All pairs (temperature, humidity), with the same id such that humidity is the first humidity measurement after temperature”
Example Pattern with Kleene Closure

Plantations might be at risk if sensor 1 detects low temperatures after a high-humidity period

PATTERN SEQ(H+ h[], T t)
WHERE h.id = 1 AND t.id = 1
AND h.hum > 70 AND t.temp < 10
WITHIN 5 minutes
HAVING count(h) = 4

Plantations might be at risk if sensor 1 detects low temperatures after a high-humidity period

T1: H
T1.id = 1
T1.hum > 70

T2: H
T2.id = 1
T2.hum > 70

T3: T
T3.id = 1
T3.temp < 10

T3.ts – T1.ts ≤ 5min

count = 3
Semantics of KSEQ

KSEQ
- Semantics given by evaluation procedure
- In ZStream: Window of length \textit{count} slides over buffer

\begin{align*}
\text{T1: } & \text{H} & \text{T2: } & \text{H} & \text{T3: } & \text{T} \\
\text{T1.id = 1} & \text{T2.id = 1} & \text{T3.id = 1} \\
\text{T1.hum > 70} & \text{T2.hum > 70} & \text{T3.temp < 10} \\
\text{count = 3} & & & \text{T3.ts - T1.ts \leq 5min} \\
\end{align*}

\begin{tabular}{ccccccccc}
\text{h} & \text{t} & \text{h} & \text{h} & \text{t} & \text{h} & \text{h} & \text{h} & \text{t} \\
75 & 35 & 78 & 80 & 20 & 79 & 76 & 8 & 73 & 8 & 6 & 68 & 4 \\
\end{tabular}
How to use tree-based models for event recognition?
Event Recognition in ZStream

Main idea

- Each node is assigned an event buffer
- Events in buffers are ordered by timestamp
- Incoming events are put into leaf buffers
- Actual recognition is realised in iterations:
  - Idle rounds: Events are only inserted into leaf buffers
  - Assembly rounds: Evaluation of operators of non-leaf nodes

Major advantage

- Intermediate events are assembled in a lazy manner
ZStream’s Batch Iterator Model

Process incoming events in batches
  – To increase efficiency
  – To increase robustness with respect to out-of-order arrival

General procedure
  1) Read batch of events into leaf buffers
  2) If there is no event in final node of tree, read next batch
  3) Otherwise, calculate *earliest allowed timestamp (EAT)*:
     time of event in buffer of final node – time window
  4) Filter buffers of the tree based on EAT
  5) Assemble events from the leaves to root of the tree
Example

EVENT SEQ(T t, !P p, H h)
WHERE t.id = 1 AND p.id = 1
AND h.id = 1
WITHIN 5

T3.ts – T1.ts ≤ 5sec

SEQ

NSEQ

T1: T
T1.id = 1

T2: P
T2.id = 1

T3: H
T3.id = 1

p  t  t  p  t  p  t  t  h  h  t
id 1  1  1  1  2  3  1  2  1  1  1
ts 1  1  3  4  5  6  6  6  7  12 13
Evaluation of Negation Operator

Challenges:
- Occurrence of event may invalidate intermediate composite events
- Filter-last approach induces large number of intermediate results

Approach:
- Iterate over right child buffer (C), filter based on EAT
- Determine valid interval for each event in right child buffer (C) based on left child buffer (B)
- Buffer of NSEQ contains for each event of C, the last event of B that negates it (or no event of B)
- Timestamp of B events in buffer of NSEQ is used by parent operator when assembling events
Example

1) EAT derived from 7 is 2
2) Filter 1
3) Valid interval of 7 is [4,7], valid interval of 12 is [7,12]

T3.ts – T1.ts ≤ 5sec
T1.ts ≥ T2.ts

T1: T
T1.id = 1
T1.ts = 1, 3, 6

T2: P
T2.id = 1
T2.ts = 1, 4

T3: H
T3.id = 1
T3.ts = 7, 12

p t t p t t p t t h h t
id 1 1 1 1 2 3 1 2 1 1 1
ts 1 1 3 4 5 6 6 6 7 12 13
Evaluation of Sequence Operator

Iterate over right and left child buffer, filtering based on EAT
Check predicates for remaining events

1) EAT is 2
2) Filter
3) Assemble

T3.ts – T1.ts ≤ 5sec
T1.ts ≥ T2.ts

SEQ

T1: T
T1.id = 1

T2: P
T2.id = 1

T3: H
T3.id = 1

NSEQ

6 7

1 3 6

4

7 12
Plan Optimisation

**Opportunity:** Tree-based plans may be semantically equivalent, but differ in their efficiency
- Algebraic rewriting of trees
- Tuning of the evaluation of predicates
- Nesting/ordering of operators

**Approach:**
- Derive best *logical* plan by evaluating equivalent candidate plans obtained by rewriting rules
- Derive best *physical* plan by determining the most efficient operator order

Evaluation of plans is driven by *cost model*
Cost Model

Cost is CPU cost, as events reside in memory

Cost of a tree-based plan is the sum of

- Cost of accessing the input data:
  \#input events touched

- (Weighted) cost of predicate evaluation:
  \#predicates \times \#input events touched

- (Weighted) cost of assembling the output data:
  \#events created
Cost of Sequence Operator

Cardinality of events of type X that are active in time window, estimated as:

\[ C_X = \text{rate}_X \times \text{window} \times \text{type selectivity} \]

\[ C_{T1} = 10e/\text{sec} \times 5\text{sec} \times \frac{1}{10} = 5e \]

\[ C_{T2} = 20e/\text{sec} \times 5\text{sec} \times \frac{1}{5} = 20e \]

Cost of accessing the input data:

\[ C_{\text{In}} = C_{T1} \times C_{T2} \times \text{ordering selectivity} \]

\[ = 5e \times 20e \times \frac{1}{2} = 50e^2 \]

Cost of accessing the output data:

\[ C_{\text{Out}} = C_{\text{In}} \times \text{correlation selectivity} \]

\[ = 50e^2 \times \frac{1}{5} = 10e^2 \]
Plan Transformations

**Idea:** Rewriting rules preserve semantics

- Fixed set of rules that is applicable
- Outcome of applying a rule is assessed by cost model
- Heuristic search instead of optimal solution

```
PATTERN SEQ(A a, (!B b UNION !C c), D d)
```

```
PATTERN SEQ(A a, !(B b INTERSECT C c), D d)
```

```
SEQ
```

```
NSEQ
```

```
DIS
```

```
A
```

```
B
```

```
C
```

```
D
```
Plan Reordering

Idea: Obtain most efficient operator nesting
- Nesting problem has optimal substructure
- Bottom-up approach, starting with optimal plan for pairs of event types

EVENT SEQ(T t, H h1, H h2)
Summary Tree-based Models

Trees of event operators...
.... as a model to define composite events
.... as a model to conduct event recognition

Enables optimisations of physical plan
  – Incorporate selectivities
  – Plan may be changed dynamically

Open: Comparison to automata-based models
Part 3: Logic-based models and methods
Logic-based models: overview

- Mainly used for modeling
- *Declarative* approach
  - Specifies *what* not *how*
- Formal semantics of CER based on logic
Logic-based models: overview

- Mainly used for modeling
- *Declarative* approach
  - Specifies *what* not *how*
- Formal semantics of CER based on logic
- Different variants
  - Support for static background knowledge
  - Support for dynamic changes in the background knowledge
  - Support for durative events
  - Support for metric temporal constraints
  - Support for out-of-order events
  - ...
Logic-based models: overview

- Various approaches to event recognition
  - Automata-based approaches
  - Tree-based approaches
  - Lazy evaluation
  - ...

Logic-based models: outline

- Chronicle Recognition System
- TESLA / T-Rex
- Real-Time Event Calculus (RTEC)
- ETALIS

Main references:


G. Cugola and A. Margara. *TESLA: a formally defined event specification language*. DEBS 2010


CE as Chronicle

- **Chronicle**: a set of events interlinked by time constraints and whose occurrence may depend on the context
  - Chronicles are used to define CEs
Chronicle Recognition System

- CRS is based on reified temporal logic
- Discrete time
- Events + properties (to represent context information)
  - They can have parameters
- Predicates for event occurrence, absence, repetition, ...
## Chronicle Representation Algebra

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>event(E, T)</td>
<td>Event E takes place at time T</td>
</tr>
<tr>
<td>event(F: (?V1, ?V2), T)</td>
<td>An event takes place at time T changing the value of property F from ?V1 to ?V2</td>
</tr>
<tr>
<td>noevent(E, (T1,T2))</td>
<td>Event E does not take place between [T1, T2)</td>
</tr>
<tr>
<td>noevent(F: (?V1, ?V2), (T1,T2))</td>
<td>No event takes place between [T1, T2) that changes the value of property F from ?V1 to ?V2</td>
</tr>
<tr>
<td>hold(F: ?V, (T1,T2))</td>
<td>The value of property F is ?V between [T1, T2)</td>
</tr>
<tr>
<td>occurs(N,M,E, (T1,T2))</td>
<td>Event E takes place at least N times and at most M times between [T1,T2)</td>
</tr>
</tbody>
</table>
Chronicle Representation Language

```plaintext
chronicle abnormal_vessel_movement[?id](T2) {
  event( speedChange[?id], T0 )
  event( speedChange[?id], T1 )
  event( speedChange[?id], T2 )
  T1 > T0
  T2 > T1
  T2 - T0 in [1, 20000]
  noevent( turn[?id], ( T0+1, T2 ) )
}
```
Chronicle Representation Language

- Purely temporal reasoning: mathematical operators in the atemporal constraints of the language are not allowed
  - No spatial reasoning (distance)
  - No use of background knowledge
Chronicle Representation Language

- Purely temporal reasoning: mathematical operators in the atemporal constraints of the language are not allowed
  - No spatial reasoning (distance)
  - No use of background knowledge
- Universal quantification is not allowed
  - Cannot express a property about all vessels in some area
Pure temporal reasoning makes CRS efficient

Each CE definition is represented as a Temporal Constraint Network, as below
Chronicle Recognition System: Compilation

- Constraint propagation in the Temporal Constraint Network
  - Build the least constrained network
- Consistency checking
  - Detect inconsistencies at compile time
Chronicle Recognition System: Recognition

Recognition stage
- Partial CE instance evolution
- Forward (incremental) recognition
Chronicle Recognition System: Partial instances

A \rightarrow^ {[1,3]} B \rightarrow^ {[0,3]} C

time

duplicated
Chronicle Recognition System: Partial instances

A@1

A → B
[1,3]

B → C
[0,3]

time

A@3

B@5

killed instance

C@3

B@5

duplicated

C@5
Chronicle Recognition System: Partial instances

\[ A \rightarrow B^{[1,3]} \rightarrow C^{[0,3]} \]

\[ A@1 \rightarrow B^{[2,4]} \]

\[ A@3 \rightarrow B^{[5,6]} \rightarrow C^{[5,8]} \]

\[ A@3 \rightarrow B^{[4,6]} \]

\[ time \]
Chronicle Recognition System: Partial instances

A [1,3] B [0,3] C

A@1 A@3 A@1

duplicated

A@1 B[2,4]
Chronicle Recognition System: Partial instances

A → B → C

[1,3] [0,3]

A@1

A@3

A@1 → B[2,4]

A@3 → B[3,4]

A@1 → B[3,4]

A@3 → B[4,6]

killed instance
Chronicle Recognition System: Partial instances

A → [1,3] → B → [0,3] → C

A@1 ——— A@3 ——— B@5

A@1 → B[2,4]
A@1 → B[3,4]
A@1 → B[3,4]
A@3 → B[4,6]
Chronicle Recognition System: Partial instances

A → B \([1,3]\) → C

A@1 \(\rightarrow\) B[2,4]

A@3 \(\rightarrow\) B[3,4]

B@5 \(\rightarrow\) time

A@1 \(\rightarrow\) B[3,4]

A@3 \(\rightarrow\) B[4,6]

killed instance
Chronicle Recognition System: Partial instances

A [1,3] B [0,3] C

A@1 A@3 B@5 killed instance
duplicated

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

A@3 B@5 time
Chronicle Recognition System: Partial instances

A@1 \rightarrow B[2,4]
A@3 \rightarrow B[3,4] \text{ killed instance}
A@3 \rightarrow B[4,6] \rightarrow B[5,6] \rightarrow C[5,8]
A@3 \rightarrow B@5 \rightarrow C[5,8]

A \rightarrow [1,3] B \rightarrow [0,3] C

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

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
Chronicle Recognition System

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
Chronicle Recognition System

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.
- The performance of CRS depends directly on the number of partial CE instances.
- To deal with out-of-order events, CRS keeps in memory partial CE instances longer.
Chronicle Recognition System: Summary

- One of the earliest and most successful formal event processing systems
- Many of its features appear in modern event processing systems
- Efficient and scalable event recognition
Chronicle Recognition System: Summary

- One of the earliest and most successful formal event processing systems
- Many of its features appear in modern event processing systems
- Efficient and scalable event recognition
- **But**: it is a purely temporal reasoning system
Define Fire(area: string, temp: double)
From Humidity(percentage < 25 and area = $a$) and last Temp(value > 40 and area = $a$) within 5 min from Humidity
where area = Temp.area, temp = Temp.value
Define Fire(area: string, temp: double)
From Humidity(percentage \(<\) 25 and area = $a$) and last Temp(value \(>\) 40 and area = $a$) within 5 min from Humidity
where area = Temp.area, temp = Temp.value

Combine Humidity only with the most recent Temp that satisfies the constraints
Define Fire(area: string, temp: double)
From Humidity(percentage < 25 and area = $a$) and each Temp(value > 40 and area = $a$)
within 5 min from Humidity
where area = Temp.area, temp = Temp.value

Combine Humidity with all Temp in the window of 5 minutes that satisfy the constraints
Define Fire(area: string, temp: double)
From Humidity(percentage < 25 and area = $a$) and last Temp(value > 40 and area = $a$) within 5 min from Humidity where area = Temp.area, temp = Temp.value consuming Temp

Temp is not available for further triggering of the rule
TESLA: Semantics

- All the operators have formal semantics
  - Based on metric temporal logic formulas
- Temporal patterns
  - Starting from an anchor point that determines when the complex event occurs
  - Temporal relations that specify when –*in the past*– other events must occur
TESLA / T-Rex: Delayed Processing

- Initial implementation based on automata
  - AIP: Automata Incremental Processing
- Second (current) version based on *lazy evaluation*
  - CDP: Column-based Delayes Approach
  - Similar to the *temporal focusing* optimization in CRS
  - Always wait for an event that might terminate a valid sequence
TESLA / T-Rex: Delayed Processing
TESLA / T-Rex: Delayed Processing

A → B [0,5] → B → C [0,3]
TESLA / T-Rex: Delayed Processing

A \rightarrow [0,5] \rightarrow B \rightarrow [0,3] \rightarrow C

\begin{array}{c}
A@1 \\
A@2 \\
A@4 \\
A@6 \\
B@3 \\
B@8 \\
C@9
\end{array}
TESLA / T-Rex: Delayed Processing
TESLA / T-Rex: Delayed Processing

A → B → C

A@1 A@2 A@4 A@6

B@3

C@9
TESLA / T-Rex: Delayed Processing

A \rightarrow [0,5] \rightarrow B \rightarrow [0,3] \rightarrow C
TESLA / T-Rex: Delayed Processing

A \[\rightarrow\] B \[\rightarrow\] C

A@1
A@2
A@4
A@6
B@3
B@8
C@9
B@3
TESLA / T-Rex: Delayed Processing
TESLA / T-Rex: Delayed Processing
TESLA / T-Rex: Delayed Processing

A → [0,5] → B → [0,3] → C

A@1
A@2
A@4
A@6
B@3
B@8
C@9

AØ6
AØ4
AØ2
AØ1

BØ8
BØ3
CØ9
TESLA / T-Rex: Delayed Processing

\[ A \xrightarrow{[0,5]} B \xrightarrow{[0,3]} C \]

- A@1
- A@2
- A@4
- A@6
- B@3
- B@8
- C@9
TESLA / T-Rex: Delayed Processing
TESLA / T-Rex: Delayed Processing
TESLA / T-Rex: Delayed Processing

A \xrightarrow{[0,5]} B \xrightarrow{[0,3]} C
TESLA / T-Rex: Delayed Processing

- Avoid useless computations
- Avoid duplications
- Simple memory layout → Arrays
  - Cache friendly!
Figure: Each-within
TESLA / T-Rex: Delayed Processing

Figure: Last-within
TESLA / T-Rex: Delayed Processing

- Avoid useless computations
- Avoid duplications
- Simple memory layout $\rightarrow$ Arrays
  - Cache friendly!
- Suitable for data parallelism
TESLA / T-Rex: Delayed Processing

![Graph showing delayed processing time vs size of windows for different methods: AIP Each-Within, CDP CPU Each-Within, CDP GPU Each-Within, AIP Last-Within, CDP CPU Last-Within, CDP GPU Last-Within.](image)
TESLA / T-Rex: Summary

- Formal semantics based on metric temporal logic formulas
- Flexible event selection and consumption policies
- Efficient and scalable event recognition
  - Support for parallel architectures
  - **But:** does not support out-of-order events
- **But:** does not support reasoning on background knowledge
TESLA / T-Rex: Summary

- Formal semantics based on metric temporal logic formulas
- Flexible event selection and consumption policies
- Efficient and scalable event recognition
  - Support for parallel architectures
  - But: does not support out-of-order events
- But: does not currently support reasoning on background knowledge
TESLA / T-Rex: Summary

- Formal semantics based on metric temporal logic formulas
- Flexible event selection and consumption policies
- Efficient and scalable event recognition
  - Support for parallel architectures
  - But: does not support out-of-order events
- But: does not support reasoning on background knowledge
- But: does not support durative events
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
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 (RTEC)

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>happensAt(E, T)</code></td>
<td>Event $E$ occurs at time $T$</td>
</tr>
<tr>
<td><code>initiatedAt(F = V, T)</code></td>
<td>At time $T$ a period of time for which $F = V$ is initiated</td>
</tr>
<tr>
<td><code>terminatedAt(F = V, T)</code></td>
<td>At time $T$ a period of time for which $F = V$ is terminated</td>
</tr>
<tr>
<td><code>holdsFor(F = V, I)</code></td>
<td>$I$ is the list of the maximal intervals for which $F = V$ holds continuously</td>
</tr>
<tr>
<td><code>holdsAt(F = V, T)</code></td>
<td>The value of fluent $F$ is $V$ at time $T$</td>
</tr>
<tr>
<td><code>union_all([J₁, ..., Jₙ], I)</code></td>
<td>$I = (J₁ \cup ... \cup Jₙ)$</td>
</tr>
<tr>
<td><code>intersect_all([J₁, ..., Jₙ], I)</code></td>
<td>$I = (J₁ \cap ... \cap Jₙ)$</td>
</tr>
<tr>
<td><code>relative_complement_all(I', [J₁, ..., Jₙ], I)</code></td>
<td>$I = I' \setminus (J₁ \cup ... \cup Jₙ)$</td>
</tr>
</tbody>
</table>
CE Definitions in RTEC: Simple Fluents

**CE definition:**

\[
\text{initiatedAt}(CE, T) \leftarrow \text{happensAt}(E_{I_{n_1}}, T), \quad \text{[conditions]}
\]

\[
\cdots
\]

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

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

\[
\cdots
\]

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

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:

\[
\begin{align*}
\text{initiatedAt}(CE, T) & \leftarrow \\
\text{happensAt}(E_{ln1}, T), & \quad \text{[conditions]} \\
\cdots & \\
\text{initiatedAt}(CE, T) & \leftarrow \\
\text{happensAt}(E_{lni}, T), & \quad \text{[conditions]} \\
\text{terminatedAt}(CE, T) & \leftarrow \\
\text{happensAt}(E_{T1}, T), & \quad \text{[conditions]} \\
\cdots & \\
\text{terminatedAt}(CE, T) & \leftarrow \\
\text{happensAt}(E_{Tj}, T), & \quad \text{[conditions]}
\end{align*}
\]

CE recognition

![Diagram of time with events]
CE Definitions in RTEC: Simple Fluents

CE definition:

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

\[
\ldots
\]

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

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

\[
\ldots
\]

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

CE recognition:

\[
\begin{array}{c}
\text{time} \\
0
\end{array}
\]
CE Definitions in RTEC: Simple Fluents

CE definition:

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

CE recognition:

\[\begin{array}{c}
\text{time} \\
\hline
0 \\
\hline
\end{array}\]
CE Definitions in RTEC: Simple Fluents

CE definition:

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

\[
\text{...}
\]

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

\[
\text{...}
\]

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

\[
\text{...}
\]

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

CE recognition: \textit{holdsFor}(CE, I)

\[
\begin{array}{c}
\text{time} \\
0
\end{array}
\]
CE Definition in RTEC: Simple Fluents

CE definition:

\[ \text{initiatedAt}(\text{leaving.object}(P, Obj) = \text{true}, \ T) \leftarrow \]
\[ \text{happensAt}(\text{appear}(Obj), \ T), \]
\[ \text{holdsAt}(\text{inactive}(Obj) = \text{true}, \ T), \]
\[ \text{holdsAt}(\text{close}(P, Obj) = \text{true}, \ T), \]
\[ \text{holdsAt}(\text{person}(P) = \text{true}, \ T) \]

\[ \text{terminatedAt}(\text{leaving.object}(P, Obj) = \text{true}, \ T) \leftarrow \]
\[ \text{happensAt}(\text{disappear}(Obj), \ T) \]

CE recognition: \[ \text{holdsFor}(\text{leaving.object}(P, Obj) = \text{true}, \ I) \]
CE Definitions in RTEC: Statically Determined Fluents

\[
\text{holdsFor}(CE, I) \leftarrow \\
\text{holdsFor}(F_1, I_{F_1}), \\
\ldots, \\
\text{holdsFor}(F_f, I_{F_f}), \\
\text{interval\_manipulation}_1(I_{\alpha}, \ldots, I_{\omega}), \\
\ldots, \\
\text{interval\_manipulation}_k(I_{A}, \ldots, I_{\Omega})
\]

where

\[
\text{interval\_manipulation}(I_1, \ldots, I_n, I) :
\begin{align*}
\text{union}([I_1, \ldots, I_n], I) \\
\text{intersection}([I_1, \ldots, I_n], I) \\
\text{relative\_complement}(I_1, [I_2, \ldots, I_n], I)
\end{align*}
\]
CE Definitions in RTEC: Statically Determined Fluents

CE definition:

\[
\begin{align*}
\text{holdsFor}(abnormal(Vessel) = \text{true}, \ I) & \leftarrow \\
\text{holdsFor}(slowMotion(Vessel) = \text{true}, \ I_1), \\
\text{holdsFor}(gap(Vessel) = \text{true}, \ I_2), \\
\text{holdsFor}(stop(Vessel) = \text{true}, \ I_3), \\
\text{union}([I_1, I_2, I_3], \ I)
\end{align*}
\]
CE Definitions in RTEC: Statically Determined Fluents

CE definition:

\[
\text{holdsFor}(\text{abnormal}(\text{Vessel}) = \text{true}, \ l) \leftarrow \\
\text{holdsFor}(\text{slowMotion}(\text{Vessel}) = \text{true}, \ l_1), \\
\text{holdsFor}(\text{gap}(\text{Vessel}) = \text{true}, \ l_2), \\
\text{holdsFor}(\text{stop}(\text{Vessel}) = \text{true}, \ l_3), \\
\text{union}([l_1, l_2, l_3], \ l)
\]

Shorthand:

\[
\text{abnormal}(\text{Vessel}) \text{ iff} \\
\text{slowMotion}(\text{Vessel}) \text{ or} \\
\text{gap}(\text{Vessel}) \text{ or} \\
\text{idle}(\text{Vessel})
\]
CE Hierarchies
CE Hierarchies: Caching
CE Hierarchies: Caching
CE Hierarchies: Caching
CE Hierarchies: Caching
Run-Time Event Recognition

Real-time decision-support in the presence of

- Very large SDE streams
- Non-sorted SDE streams
- SDE revision
  - Need to retract, similar to database materialization update
- Very large CE numbers
Run-Time Event Calculus: Windowing

- Windowing improves the performance of real-time event recognition
- Event recognition repeated periodically
  - User-defined period
- At evaluation time $T$ only events that occurred in $(T - W, T]$ are considered
- Incremental algorithm with addition and retraction
  - Incremental materialization of answers
Run-Time Event Calculus: Summary

- The full power of logic programming is available
  - Complex atemporal computations
  - Combination of events streams with static knowledge
Run-Time Event Calculus: Summary

- The full power of logic programming is available
  - Complex atemporal computations
  - Combination of events streams with static knowledge
- Very efficient reasoning
  - Even when event streams arrive with a delay
  - Even in the presence of large specifications
Run-Time Event Calculus: Summary

- The full power of logic programming is available
  - Complex atemporal computations
  - Combination of events streams with static knowledge
- Very efficient reasoning
  - Even when event streams arrive with a delay
  - Even in the presence of large specifications
- **But:** The Event Calculus does not have built-in support for long-term temporal constraints
ETALIS: Overview

- As in RTEC, full power of logic programming
  - Complex atemporal computation
  - Combination of event streams with static knowledge
ETALIS: Overview

- As in RTEC, full power of logic programming
  - Complex atemporal computation
  - Combination of event streams with static knowledge
- Events have duration
  - Support for Allen’s temporal operators
ETALIS: Temporal operators
ETALIS: Example

DangerousAcceleration(X) ←
   Speed(X, Y1) SEQ Speed(X,Y2) WHERE Y2 > Y1 × 1.5

- Lack of metric temporal constraints!
ETALIS: Recognition

- Based on trees
- Binarization of operators
ETALIS: Recognition

\[ E \leftarrow A \text{ SEQ } B \text{ SEQ } C \]

Translated to:

\[ E \leftarrow E_1 \text{ SEQ } C \]
\[ E_1 \leftarrow A \text{ SEQ } B \]
ETALIS: Recognition

\[ E \leftarrow E1 \text{ SEQ } C \]
\[ E1 \leftarrow A \text{ SEQ } B \]

- Translated to Prolog rules that modify a database of facts
- When A SEQ B is detected, E1 is added to the database
ETALIS: Recognition

- Translated to Prolog rules that modify a database of facts
- When A SEQ B is detected, E1 is added to the database
- When to retract E1?
ETALIS: Recognition

E ← E1 SEQ C
E1 ← A SEQ B

- Translated to Prolog rules that modify a database of facts
- When A SEQ B is detected, E1 is added to the database
- When to retract E1?
  - Depends on the consumption policy
ETALIS: Summary

- Full power of logic programming
- Events with duration
- Complex temporal operators
- Efficient event recognition based on standard Prolog
ETALIS: Summary

- Full power of logic programming
- Events with duration
- Complex temporal operators
- Efficient event recognition based on standard Prolog
- **But:** no build-in predicate to change the value of some global properties
- **But:** no metric temporal operators
Outlook
Requirements for Complex Event Recognition Languages

- Instantaneous events.
- Durative events.
  - Open intervals.
- Context information.
- Relational events.
- No limit on the temporal distance between the events comprising a composite activity (no ‘WITHIN’ constraint).
- Concurrency constraints.
- Atemporal reasoning.
- Event hierarchies.
Outlook

There is a need for a systematic, formal comparison of:
  ▶ language expressivity;
  ▶ recognition complexity.

Some steps towards this have already been taken.

Goal: find the appropriate language (subset) to address the requirements of a given application.
Outlook for Automata

Recall:
CER automata are symbolic transducers equipped with registers.

- Is there a general evaluation strategy for this model?
- What fragments of automata can be run efficiently?
- Is there a language capturing exactly the CER patterns that can be expressed by these automata?
- What is the complexity of compiling queries into automata?
- How do different operators affect this complexity?
Outlook for Tree-based Models

**Recall:**
Trees of event operators serve as an operational model for event recognition.

- What is the language that can be represented in this model?
- What is the runtime complexity?
- How to determine whether to use tree-based models or automata?
- What about hybrid approaches that combine both types of models?
Recall:
Logic based models provide formal semantics to CER

- What is the language that can be represented in logic-based models?
- What is the relation between the different logic-based formalisms?
- What is the runtime complexity of each model?
  - How does each feature contribute to the complexity?
    - Durative events
    - Out-of-order events
    - Background knowledge
    - ...
- What is the most efficient recognition algorithm for each model?