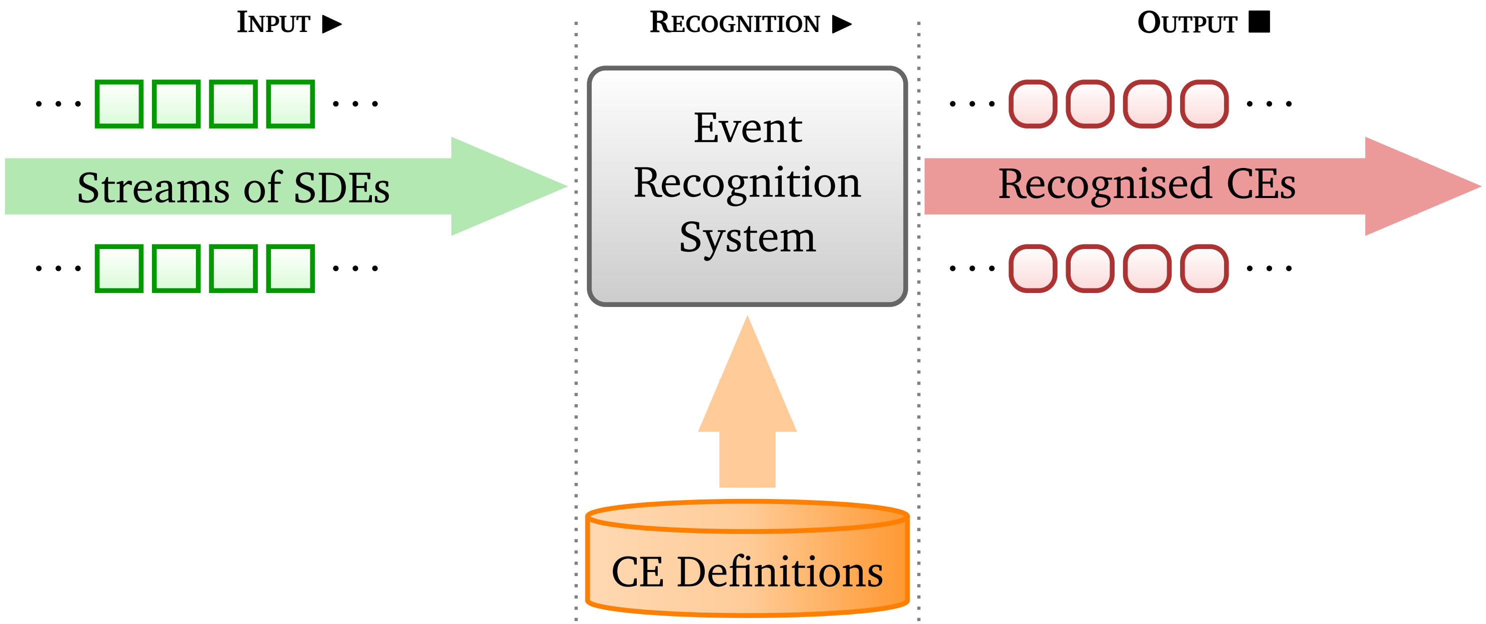
Tutorial:
Complex Event Recognition Languages

Alexander Artikis[1,2], Alessandro Margara[3], Martin Ugarte[4], Stijn Vansummeren[4], Matthias Weidlich[5]

[1]University of Piraeus, [2]NCSR Demokritos, [3]Politecnico di Milano, [4]Université Libre de Bruxelles, [5]Humboldt-Universtität zu Berlin

Complex Event Recognition (Event Pattern Matching)



Complex Event Recognition for Security

Online Recognition

Input	Output
340 $inactive(id_0)$	
340 $coord(id_0) = (20.88, -11.90)$	
340 $appear(id_0)$	
340 $walking(id_2)$	
340 $coord(id_2) = (25.88, -19.80)$	
340 $active(id_1)$	
340 $coord(id_1) = (20.88, -11.90)$	
340 $walking(id_3)$	
340 $coord(id_3) = (24.78, -18.77)$	
380 $walking(id_3)$	
380 $coord(id_3) = (27.88, -9.90)$	
380 $walking(id_2)$	
380 $coord(id_2) = (28.27, -9.66)$	

Online Recognition

Input	Output
340 $inactive(id_0)$	$since(340)$ $leaving\_object(id_1, id_0)$
340 $coord(id_0) = (20.88, -11.90)$	
340 $appear(id_0)$	
340 $walking(id_2)$	
340 $coord(id_2) = (25.88, -19.80)$	
340 $active(id_1)$	
340 $coord(id_1) = (20.88, -11.90)$	
340 $walking(id_3)$	
340 $coord(id_3) = (24.78, -18.77)$	
380 $walking(id_3)$	
380 $coord(id_3) = (27.88, -9.90)$	
380 $walking(id_2)$	
380 $coord(id_2) = (28.27, -9.66)$	

Online Recognition

Input	Output
340 $\mathit{inactive}(\mathit{id}_0)$	$\mathit{since}(340)$ $\mathit{leaving\_object}(\mathit{id}_1, \mathit{id}_0)$
340 $\mathit{coord}(\mathit{id}_0) = (20.88, -11.90)$	$\mathit{since}(340)$ $\mathit{moving}(\mathit{id}_2, \mathit{id}_3)$
340 $\mathit{appear}(\mathit{id}_0)$	
340 $\mathit{walking}(\mathit{id}_2)$	
340 $\mathit{coord}(\mathit{id}_2) = (25.88, -19.80)$	
340 $\mathit{active}(\mathit{id}_1)$	
340 $\mathit{coord}(\mathit{id}_1) = (20.88, -11.90)$	
340 $\mathit{walking}(\mathit{id}_3)$	
340 $\mathit{coord}(\mathit{id}_3) = (24.78, -18.77)$	
380 $\mathit{walking}(\mathit{id}_3)$	
380 $\mathit{coord}(\mathit{id}_3) = (27.88, -9.90)$	
380 $\mathit{walking}(\mathit{id}_2)$	
380 $\mathit{coord}(\mathit{id}_2) = (28.27, -9.66)$	

Application Requirements

- Input:
  - Instantaneous events.
  - Context information.

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.

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

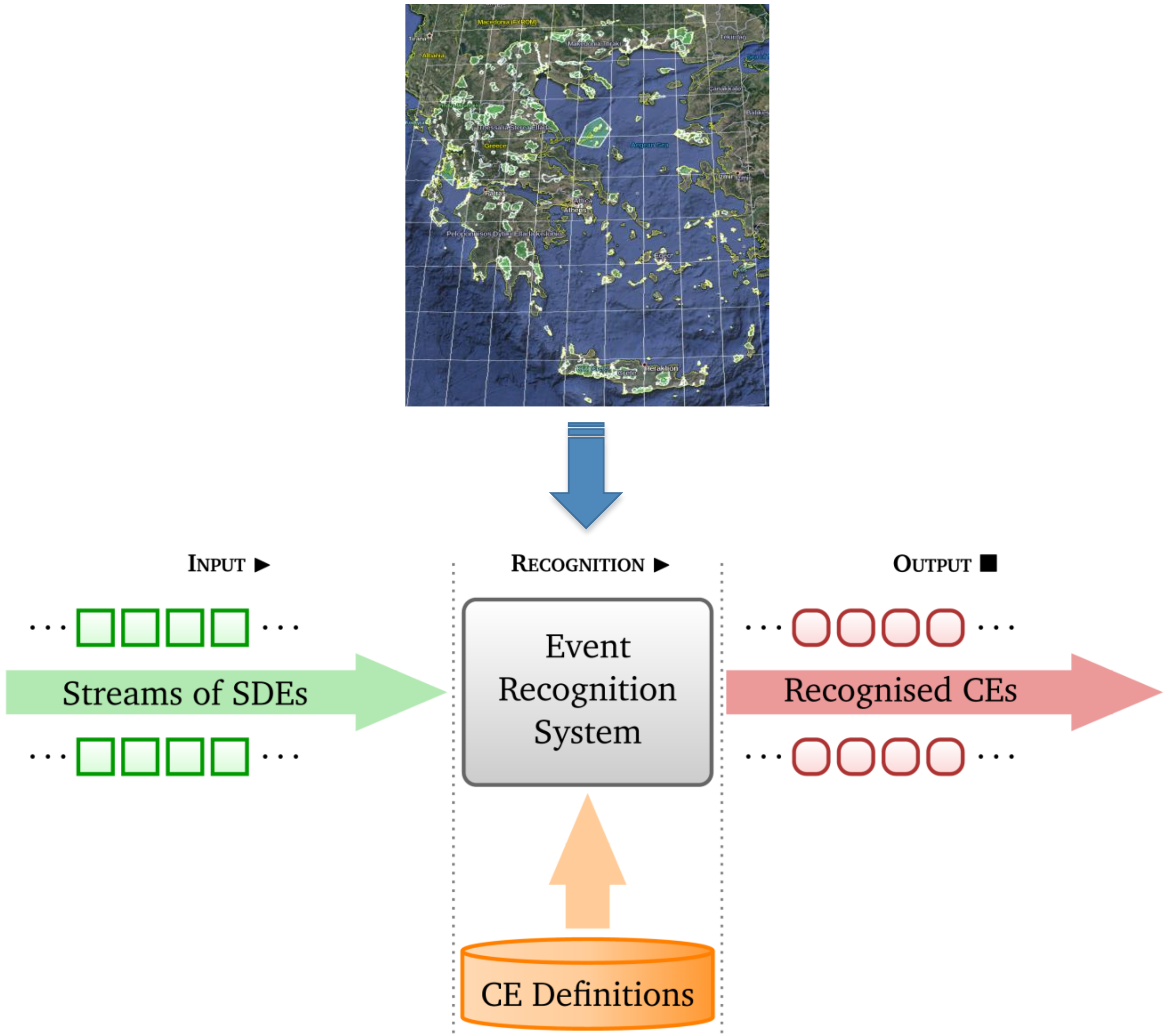
- Input:
  - Instantaneous events.
  - Context information.
- Output: durative events.
  - The interval may be open.
  - Relational events.
  - No limit on the temporal distance between the events comprising the composite activity (no 'WITHIN' constraint).
  - Concurrency constraints.
  - Spatial reasoning.

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

A symbol `t` immediately followed by a symbol **h**

An event **T** eventually followed by an event **H**

Sequencing

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

Regex `th`

h	t	t	h	h	t	h
40	35	40	25	20	45	19
1	2	3	4	5	6	7

{3,4}
{6,7}

{2,4}{3,4}
{6,7}{2,5}
{3,5}{2,7}
{3,7}

Iterations (Kleene closure)

High-humidity events $H_1, H_2, \dots, H_n$
followed by a low-temperature event T

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

?

Iterations (Kleene closure)

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

h*t

h	t	t	h	h	t	t
80	15	14	85	84	10	12
1	2	3	4	5	6	7

[1],{2}
[1],{3}
~~[1],{6}~~
~~[1],{7}~~

~~[1,4],{6}~~
~~[1,4],{7}~~
~~[1,5],{6}~~
~~[1,5],{7}~~

[1,4,5],{6}
[1,4,5],{7}
~~[4],{6}~~
~~[4],{7}~~

~~[5],{6}~~
~~[5],{7}~~
~~[4,5],{6}~~
~~[4,5],{7}~~

SEMANTICS 

CAYUGA

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

Cayuga: A General Purpose Event Monitoring System. Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, Walker M. White; CIDR 2007

Cayuga Event Language (CEL)

Named streams

Attributes

generates a stream

has

```
SELECT  <attributes>
FROM    <stream expression>
PUBLISH <output_stream_name>
```

New stream named <output_stream_name>

Projection, renaming

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)
- ...

TempIds

- T(t_id=1)
- T(t_id=2)
- T(t_id=3)
- T(t_id=1)
- T(t_id=2)
- ...

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

Filtering

Remove all tuples with temperatures below 30 degrees

```
FILTER{temp >= 30} Temp
```

Stream expression

```
SELECT    id, temp
FROM      FILTER{temp >= 30}Temp
PUBLISH   HighTemp
```

```
FILTER{<condition>}<stream_expression>
```

Stream expression

Temp

- T(id=1, temp=24)
- T(id=2, temp=35)
- T(id=3, temp=22)
- T(id=1, temp=33)
- T(id=2, temp=32)
- . . .

HighTemp

- T(id=2, temp=35)
- T(id=1, temp=33)
- T(id=2, temp=32)
- . . .

```
SELECT   id, temp
FROM     FILTER{temp >= 30}Temp
PUBLISH  HighTemp
```

`T(id, temp)`

`H(id, hum)`

Sequencing

Select the (next) temperature of areas in which a low humidity was detected

`Hum NEXT{$1.id = $2.id AND $`~~`1`~~`.hum < 25} Temp`

Condition

Stream expressions

Stream expression

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 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)

...

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 NEXT{$1
< 25} Temp
```

Compile into an automaton?

```
(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)
...
```

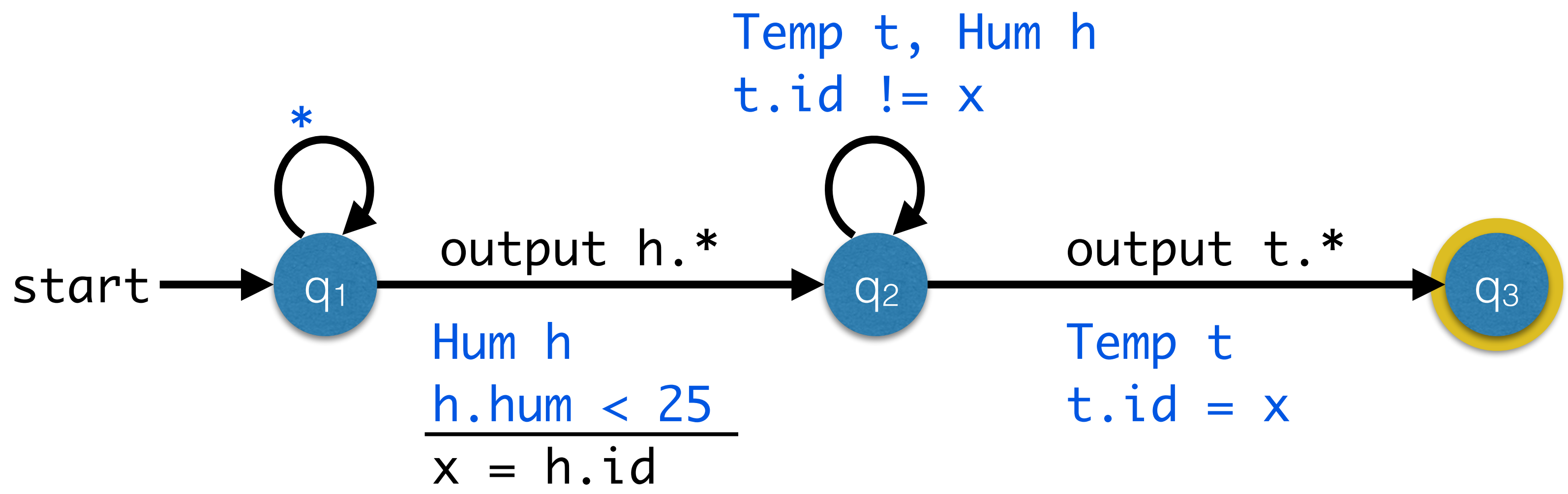
Hum

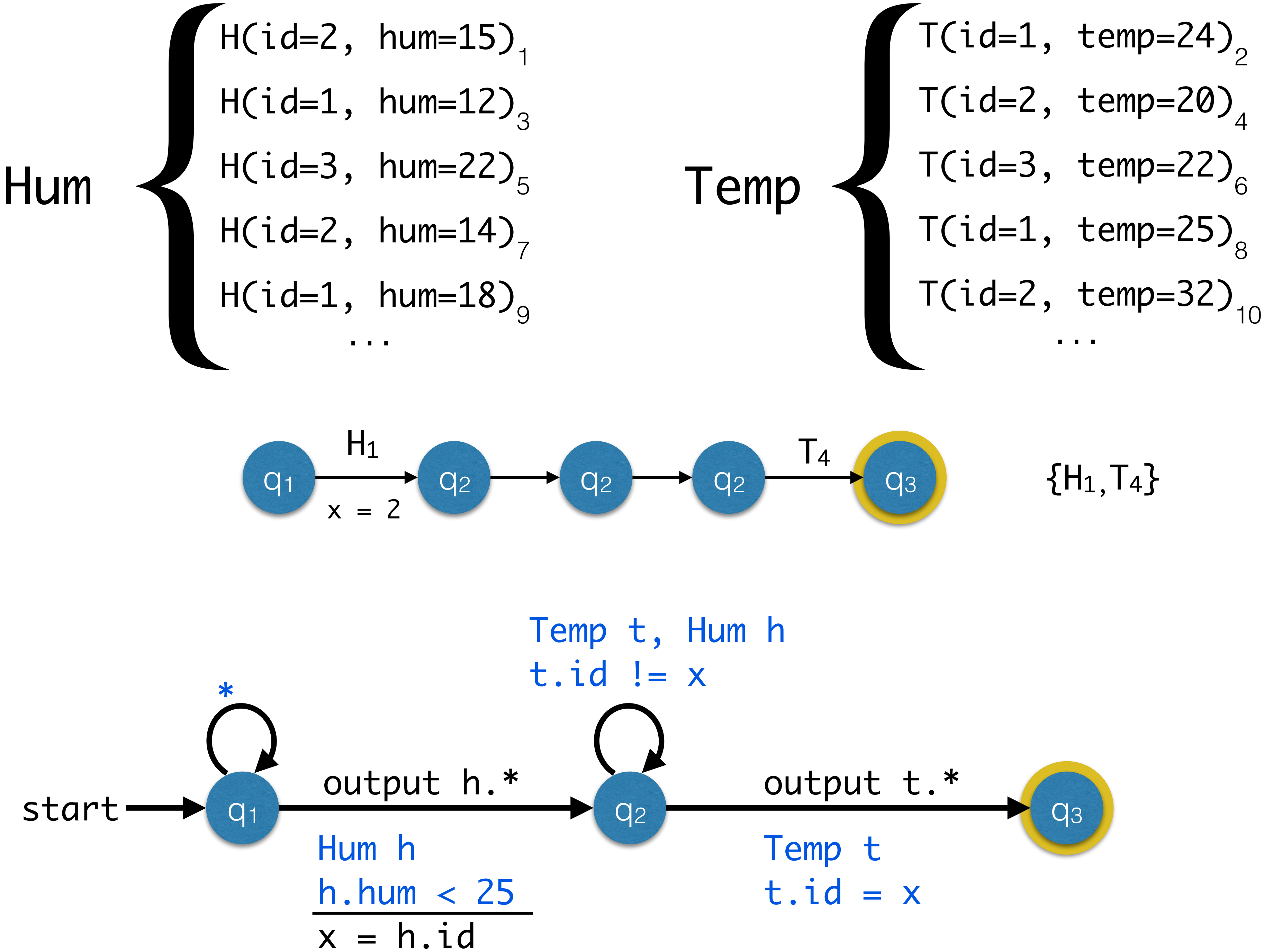
- $\texttt{H(id=2, hum=15)}_1$
- $\texttt{H(id=1, hum=12)}_3$
- $\texttt{H(id=3, hum=22)}_5$
- $\texttt{H(id=2, hum=14)}_7$
- $\texttt{H(id=1, hum=18)}_9$
- ...

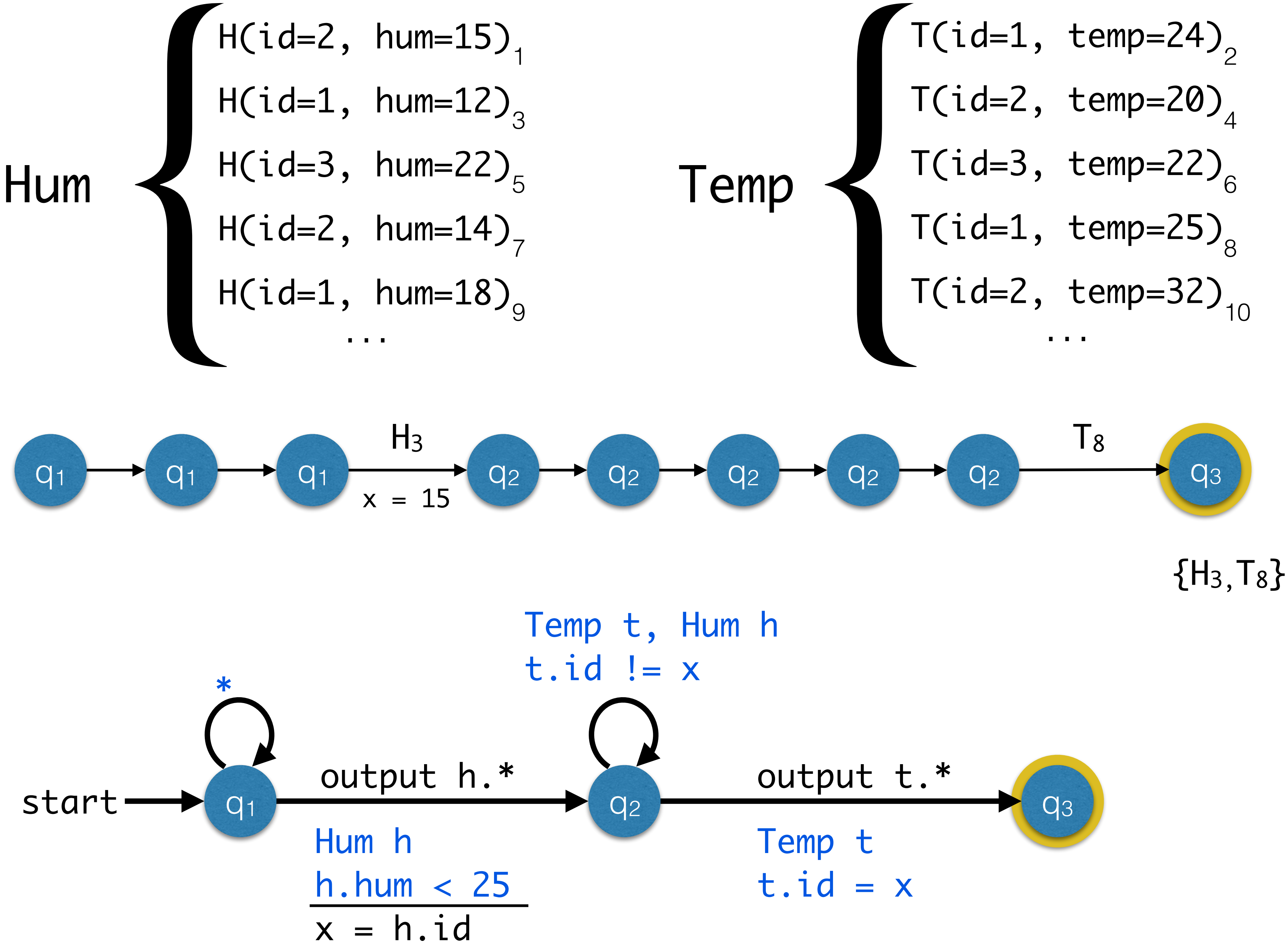
Temp

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

```
Hum NEXT{$1.id = $2.id AND $1.hum < 25} 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
...
q1
H1
x = 2
q2
q2
q2
T4
q3
{H1,T4}
start
q1
*
output h.*
Hum h
h.hum < 25
x = h.id
q2
Temp t, Hum h
t.id != x
output t.*
Temp t
t.id = x
q3



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
...
q1
q1
q1
H3
x = 15
q2
q2
q2
q2
q2
T8
q3
{H3,T8}
*
start
q1
output h.*
Hum h
h.hum < 25
x = h.id
Temp t, Hum h
t.id != x
q2
output t.*
Temp t
t.id = x
q3

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

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
)
```

(id=1, temp=24 , cnt=1)$_1$

(id=2, temp=20 , cnt=1)$_2$

(id=3, temp=19 , cnt=1)$_3$

(id=1, temp=17 , cnt=1)$_4$

(id=2, temp=18 , cnt=1)$_5$

...

1

```
$.id=1, $.temp=17, cnt=4
($1.id=1, $1.temp=24, $2.id=1, $2.temp=17, cnt=4)
```

2

```
$.id=1, $.temp=17, cnt=3
($1.id=2, $1.temp=20, $2.id=1, $2.temp=17, cnt=3)
```

3

```
$.id=1, $.temp=17, cnt=2
($1.id=2, $1.temp=20, $2.id=2, $2.temp=17, cnt=2)
```

4

```
$.id=1, $.temp=17, cnt=1
```

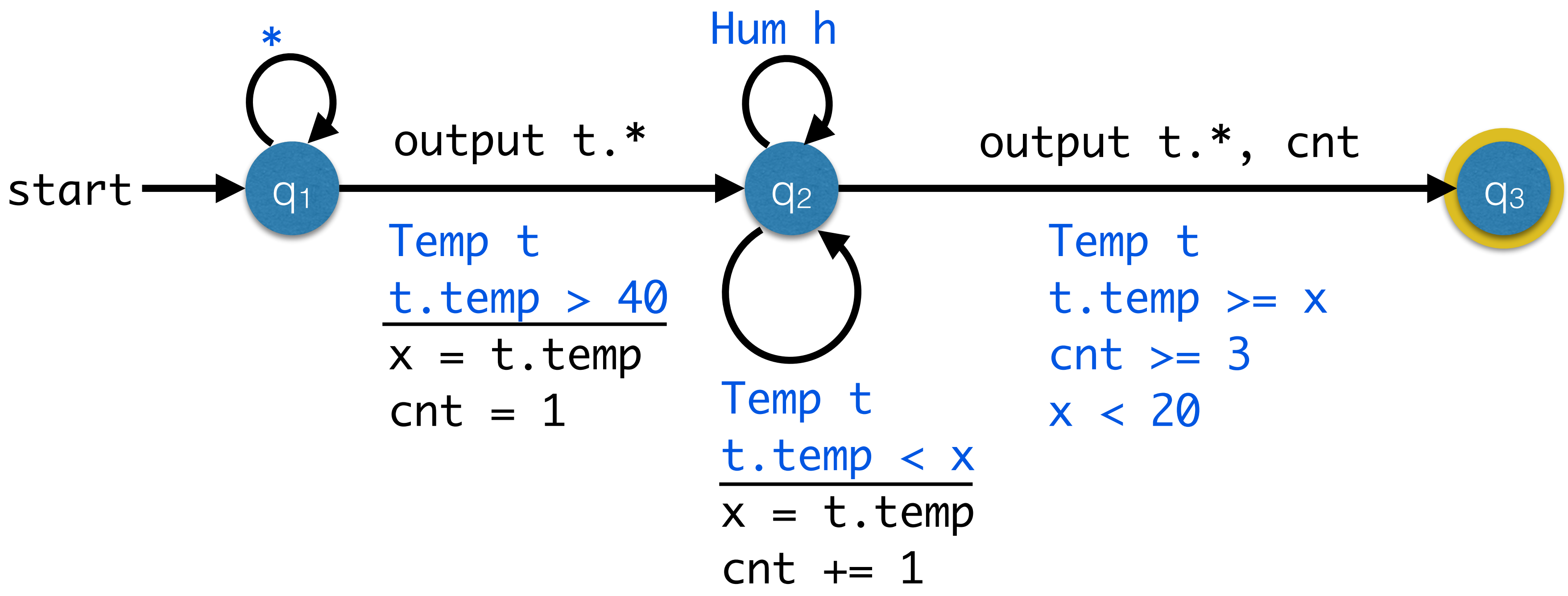
```
FILTER{cnt > 3}(
  (SELECT *, 1 AS cnt FROM Temp)
  FOLD{$.temp > $2.temp,
       $.temp <= $2.temp,
       $.cnt + 1 AS cnt } Temp
)
```

```
(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)

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

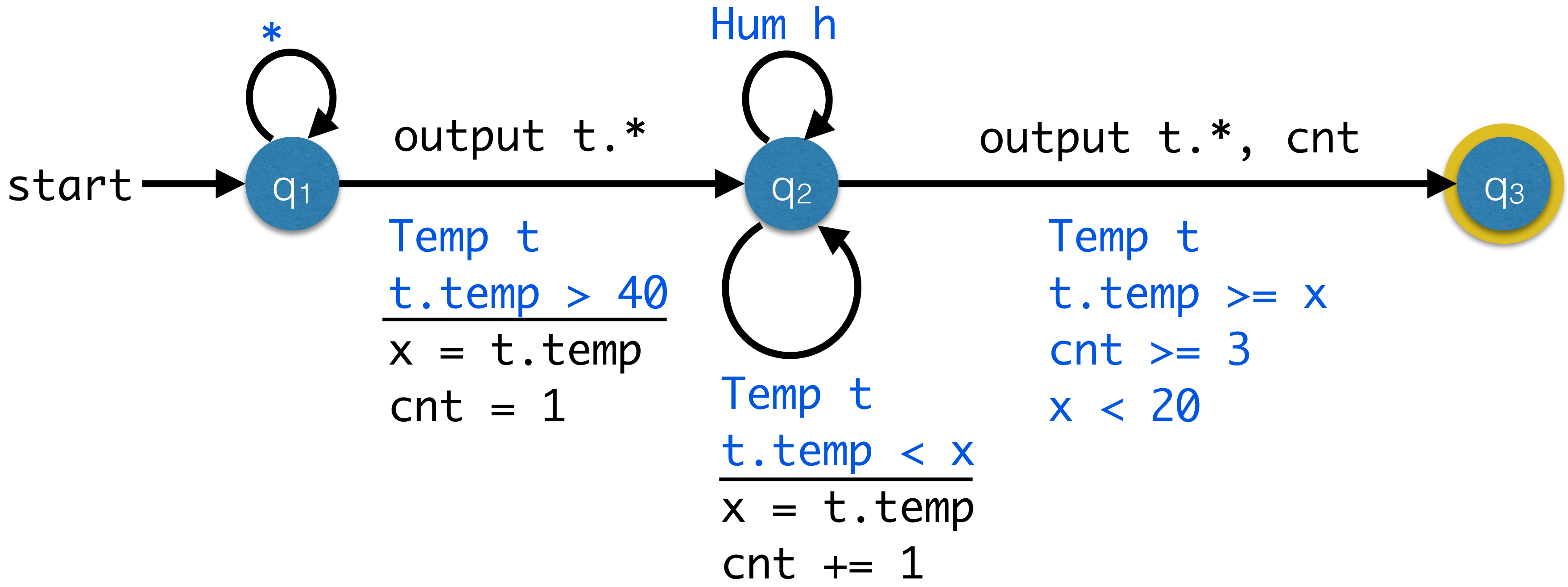
```
FILTER{cnt > 3}(
  (SELECT *, 1 AS cnt FROM Temp)
  FOLD{$.temp > $2.temp,
       $.temp <= $2.temp,
       $.cnt + 1 AS cnt } Temp
)
```

Compilation

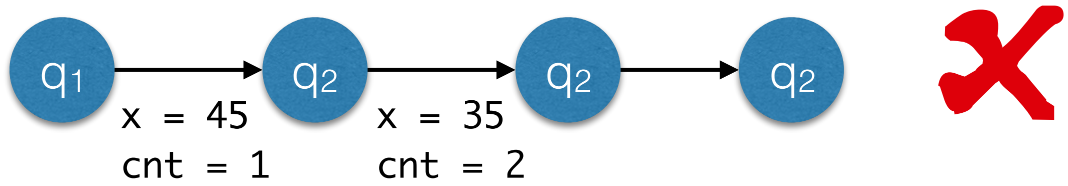
Notify me of decreasing temperature streaks of at least 3 events in which the temperature drops from above 40 to below 20

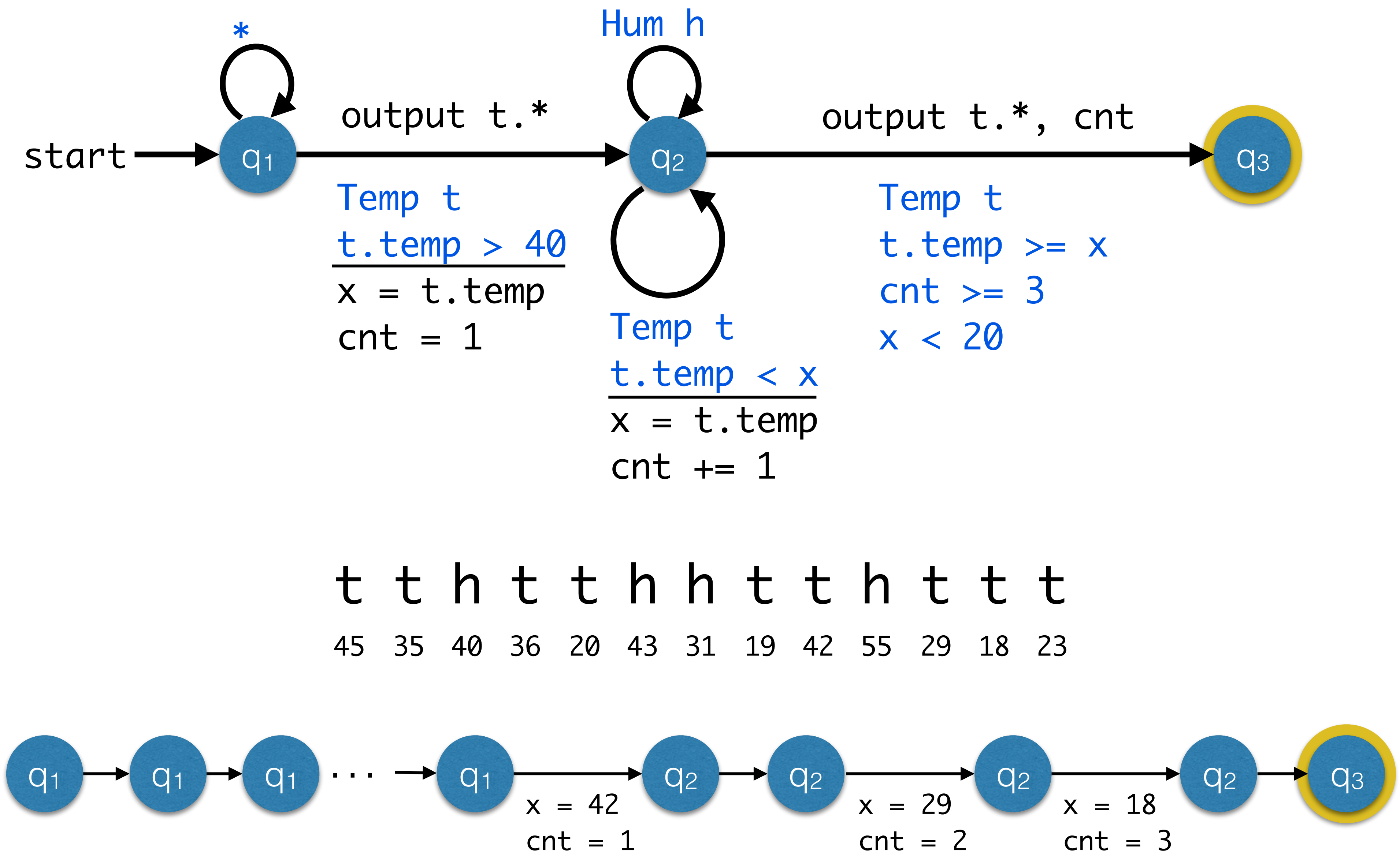



```
FILTER{cnt >= 3, temp < 20}(
   (SELECT *, 1 AS cnt FROM
                    FILTER{temp > 40}Temp)
   FOLD{$.temp > $2.temp,
        $.temp <= $2.temp OR
           (cnt >= 3 AND $.temp < 20),
        $.cnt + 1 AS cnt}
)  Temp
```



t	t	h	t	t	h	h	t	t	h	t	t	t
45	35	40	36	20	43	31	19	42	55	29	18	23





start
*
output t.*
Hum h
output t.*, cnt
q1
q2
q3
Temp t
t.temp > 40
x = t.temp
cnt = 1
Temp t
t.temp < x
x = t.temp
cnt += 1
Temp t
t.temp >= x
cnt >= 3
x < 20
t t h t t h h t t h t t t
45 35 40 36 20 43 31 19 42 55 29 18 23
q1 q1 q1 ... q1 q2 q2 q2 q2 q3
x = 42
cnt = 1
x = 29
cnt = 2
x = 18
cnt = 3

A simple (?) example

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

Will never be selected!

h	h	h	h	h	h	h	h	t	h	t	t
80	85	84	78	79	75	82	80	15	76	16	9

```
FILTER{temp < 10}(
  (SELECT *, 1 AS cnt FROM Hum)
  FOLD{$2.hum > 70,
       ($.cnt > 4 AND $2.temp < 10) OR $2.hum <= 70,
       $.cnt + 1 AS cnt}
  (Hum NEXT{True} Temp)
)
```

A simple (?) example

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

Will never be selected!

h	h	h	h	h	h	h	h	t	h	t	t
80	85	84	78	79	75	82	80	15	76	16	9

```
FILTER{temp < 10}(
  (SELEC
  FOLD{
                                              m <= 70,

  (Hum NEXT{True} Temp)
)
```

Writing rules is not a simple task…

SEMANTICS

SASE / SASE+

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

SASE+: An Agile Language for Kleene Closure over Event Streams. **Yanlei Diao, Neil Immerman and Daniel Gyllstrom; UMass Technical Report, 2007**

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

```
EVENT <event_pattern>
[WHERE <filter>]
[WITHIN <time_window>]
```

New stream, events contain <u>all attributes</u>

(no projection)

Sequencing

```
T1(id=1, temp=24, tstamp=0.5)
H1(id=2, hum=25, tstamp=0.65)
H2(id=1, hum=33, tstamp=1)
T2(id=2, temp=25, tstamp=1.32)
H3(id=2, hum=21, tstamp=1.34)
...
```

we assume that events arrive in order

```
EVENT 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(id=1, temp=24, tstamp=0.5)

H_1(id=2, hum=25, tstamp=0.65)

H_2(id=1, hum=33, tstamp=1)

T_2(id=2, temp=25, tstamp=1.32)

H_3(id=2, hum=21, tstamp=1.34)

```
EVENT SEQ(T t; H h)
WHERE t.id = h.id
```

"All pairs (`temperature`, `humidity`), with the same `id` such that `temperature` occurred before `humidity`"

{T_1, H_2},{T_2, H_3}

Time Windows

```
T1(id=1, temp=24, tstamp=0.5)
H1(id=2, hum=25, tstamp=0.65)
H2(id=1, hum=33, tstamp=1)
T2(id=2, temp=25, tstamp=1.32)
H3(id=2, hum=45, tstamp=1.34)
```

```
EVENT SEQ(T t, H h)
WHERE t.id = h.id          {T2, H3}
WITHIN 0.1 seconds
```

"All pairs (`temperature`, `humidity`), with the same `id` such that `temperature` occurred before `humidity`, but at most <u>0.1 seconds</u> before"

Negation

```
T1(id=1, temp=24, tstamp=0.5)
H1(id=2, hum=25, tstamp=0.65)
H2(id=1, hum=33, tstamp=1)
T2(id=2, temp=25, tstamp=1.32)
H3(id=2, hum=45, tstamp=1.34)
```

{T2, H3}

```
EVENT SEQ(T t, !H h1, H h2)
WHERE t.id = h2.id1
```

"All pairs (**temperature**, **humidity**), with the same **id** such that **humidity** is the <u>first</u> 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
```

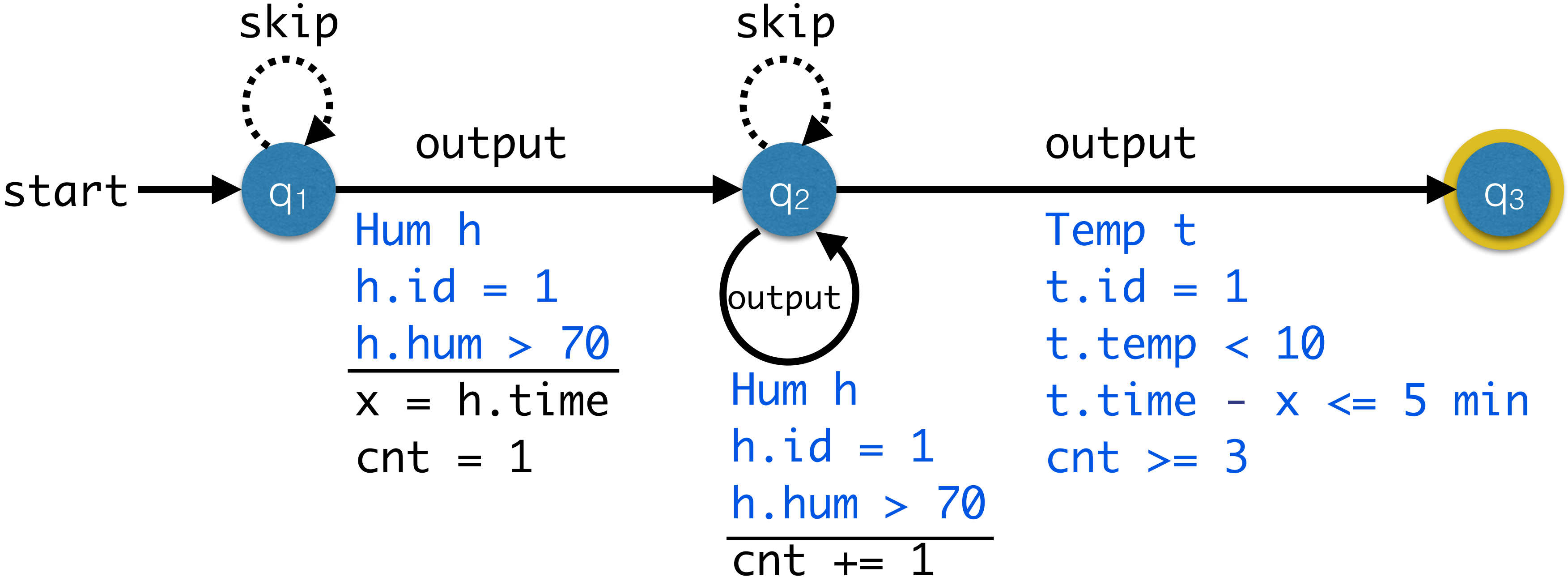
Plantations might be at risk if sensor 1 detects low temperatures after a high-humidity period

More declarative

```
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
```

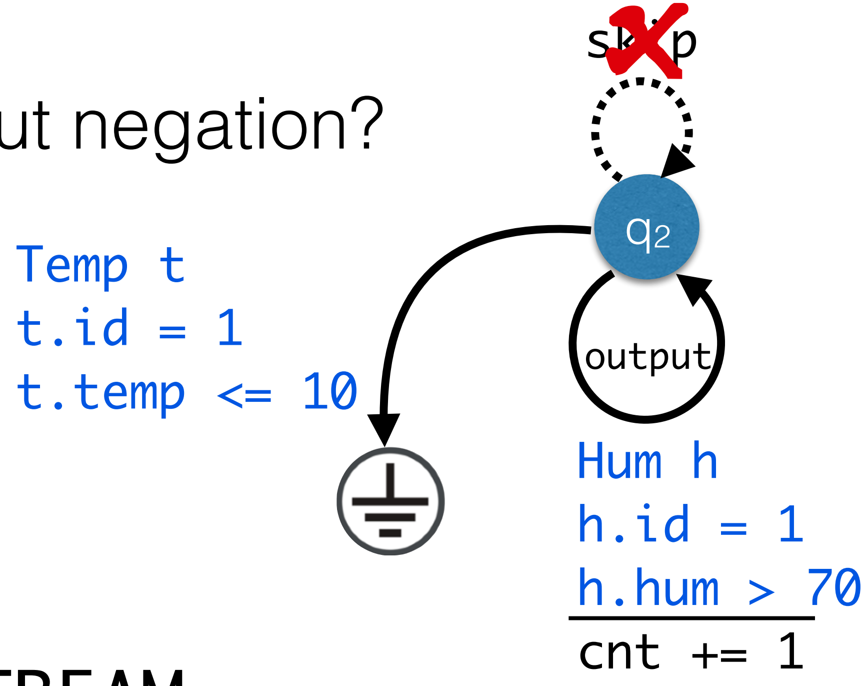
Compilation

```
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
```

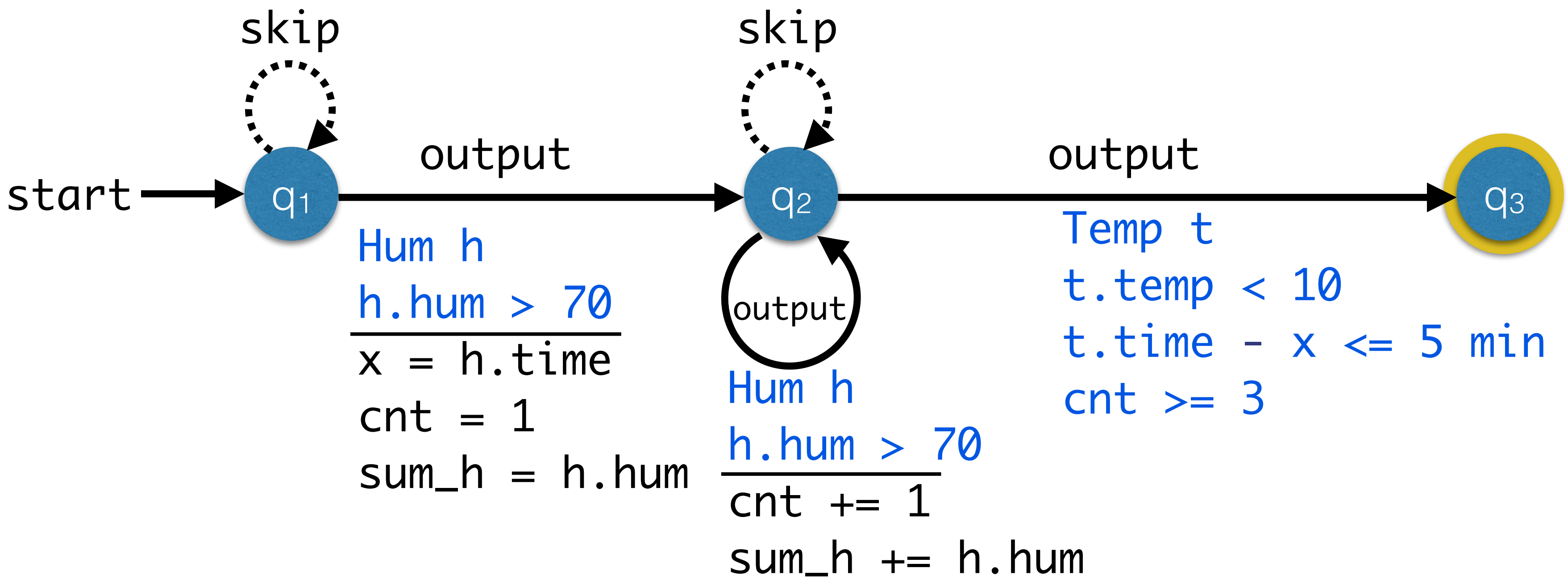
What about negation?



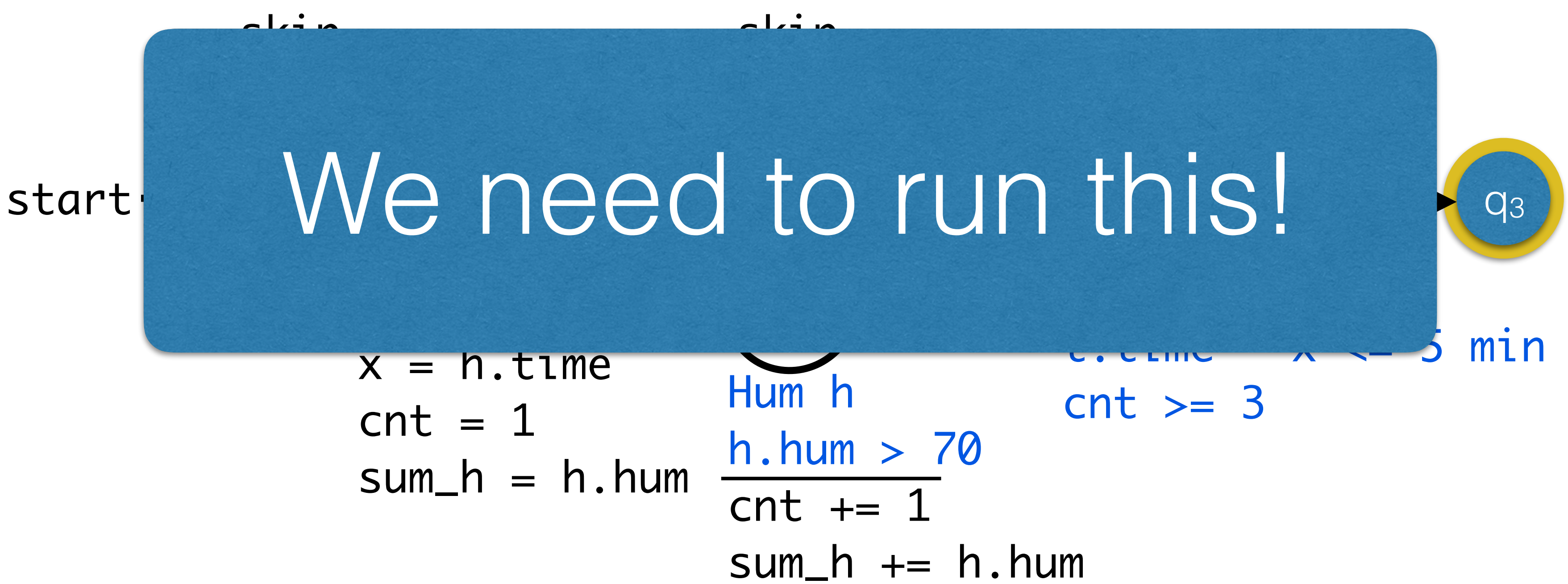

```
FROM HUM_TEMP_STREAM
PATTERN SEQ(H+ h[], !T t1, T t2)
WHERE h.id = 1 AND t1.id = 1 AND
      t2.id = 1 AND h.hum > 70 AND
      t1.temp >= 10 AND t2.temp < 10
WITHIN 5 minutes
HAVING count(h) >= 3
OUTPUT Plantation Risk
```

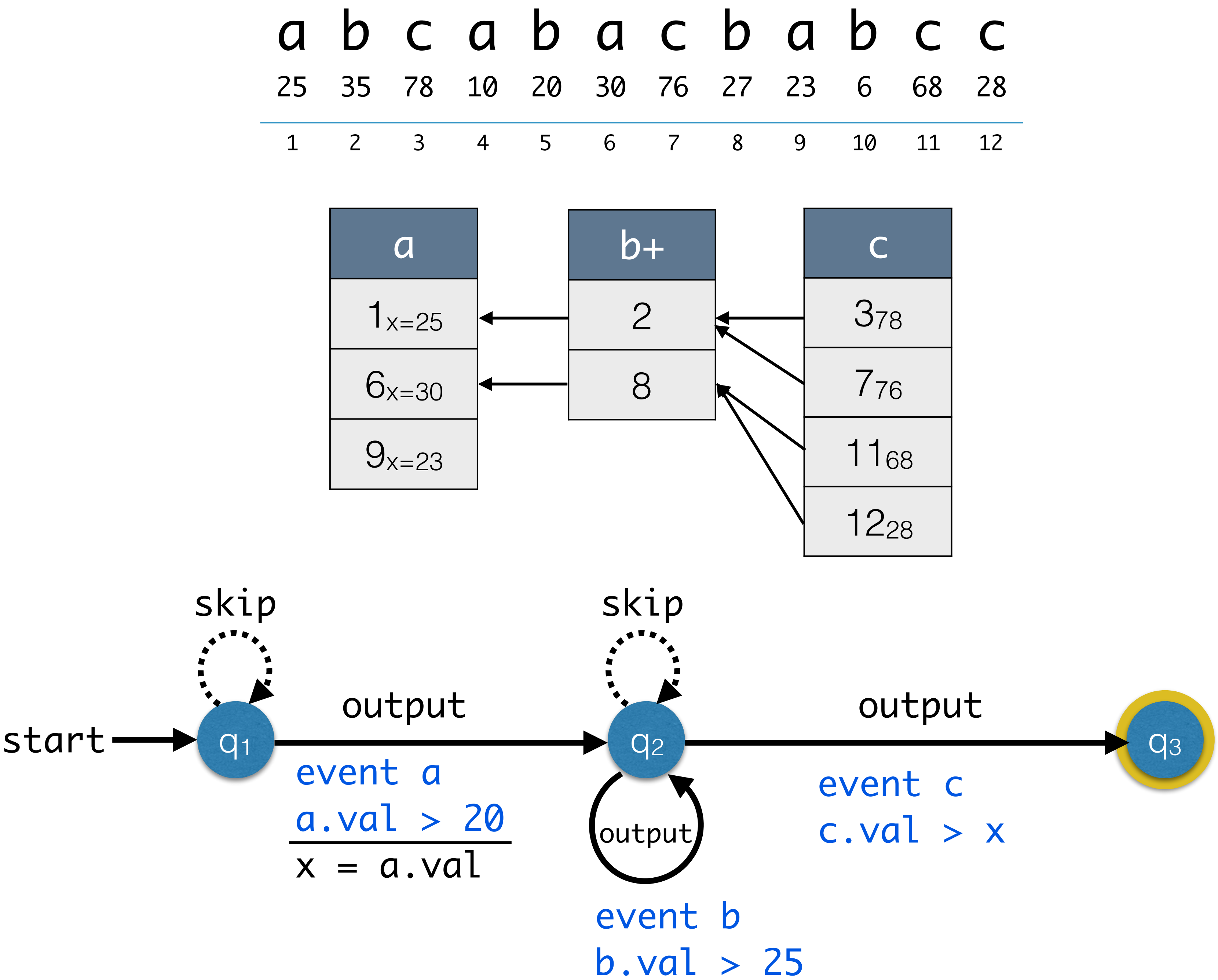
SEMANTICS

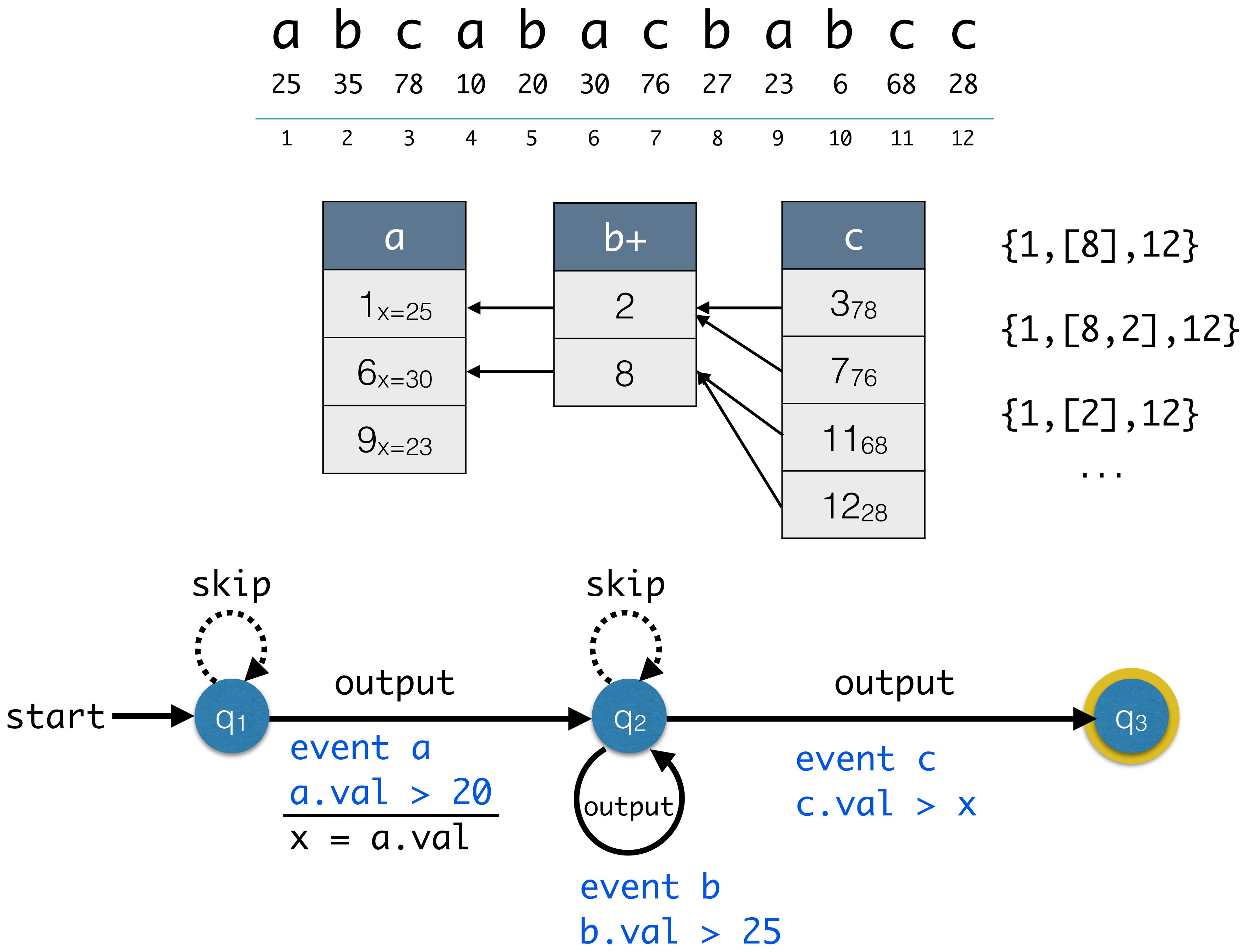
Nondeterminism?

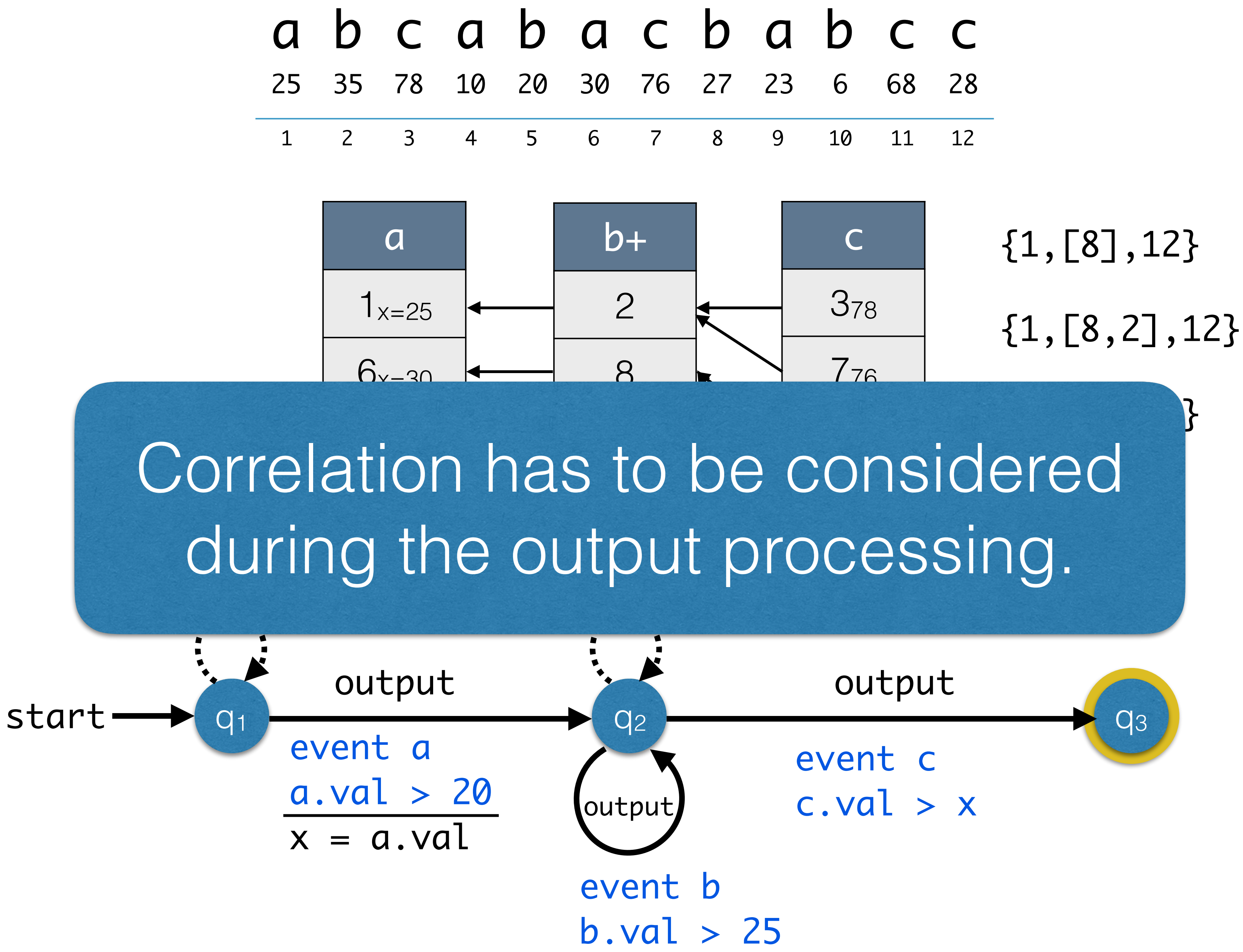


Nondeterminism?




a b c a b a c b a b c c
25 35 78 10 20 30 76 27 23 6 68 28
1 2 3 4 5 6 7 8 9 10 11 12
a
1x=25
6x=30
9x=23
b+
2
8
c
378
776
1168
1228
skip
skip
start
q1
output
q2
output
q3
event a
a.val > 20
x = a.val
output
event b
b.val > 25
event c
c.val > x


a b c a b a c b a b c c
25 35 78 10 20 30 76 27 23 6 68 28
1 2 3 4 5 6 7 8 9 10 11 12
a
1x=25
6x=30
9x=23
b+
2
8
c
378
776
1168
1228
{1,[8],12}
{1,[8,2],12}
{1,[2],12}
...
skip
skip
start
q1
q2
q3
output
output
output
event a
a.val > 20
x = a.val
event b
b.val > 25
event c
c.val > x



a b c a b a c b a b c c
25 35 78 10 20 30 76 27 23 6 68 28
1 2 3 4 5 6 7 8 9 10 11 12
a
b+
c
1x=25
2
378
{1,[8],12}
{1,[8,2],12}
Correlation has to be considered during the output processing.
start
q1
q2
q3
output
output
output
event a
a.val > 20
x = a.val
event b
b.val > 25
event c
c.val > x

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?

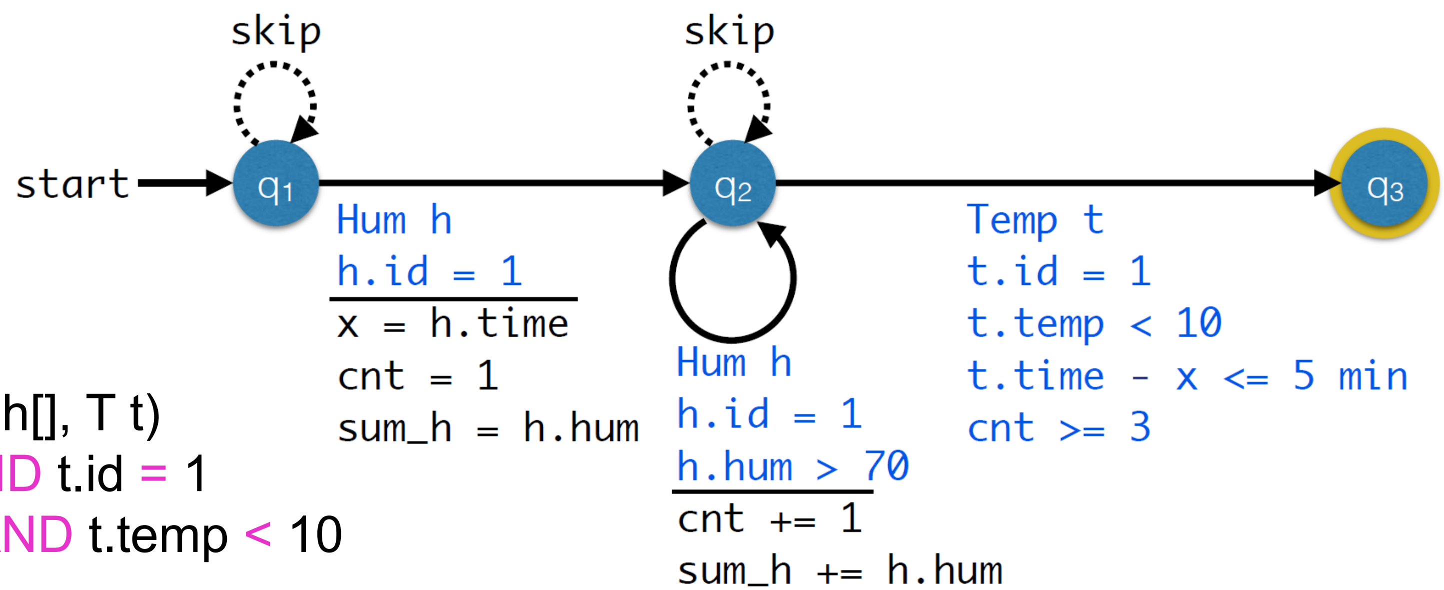
Part II

Recap of the First Part

Operators for CER

– Sequencing
– Kleene Closure
– Negation
– Filtering (predicates, time windows)

Semantics of queries given by automata

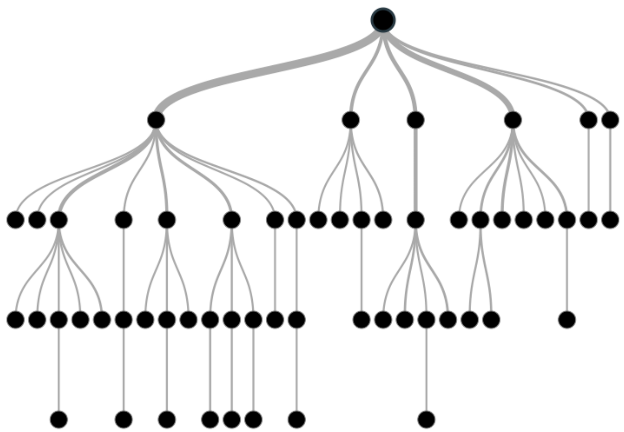



```
PATTERN SEQ(H+ h[], T t)
WHERE h.id = 1 AND t.id = 1
 AND h.hum > 70 AND t.temp < 10
WITHIN 5 minutes
HAVING count(h) >= 3
```

Tutorial Outline

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

Y. Mei and S. Madden, *ZStream: A cost-based query processor for adaptively detecting composite events*. SIGMOD 2009: 193-206

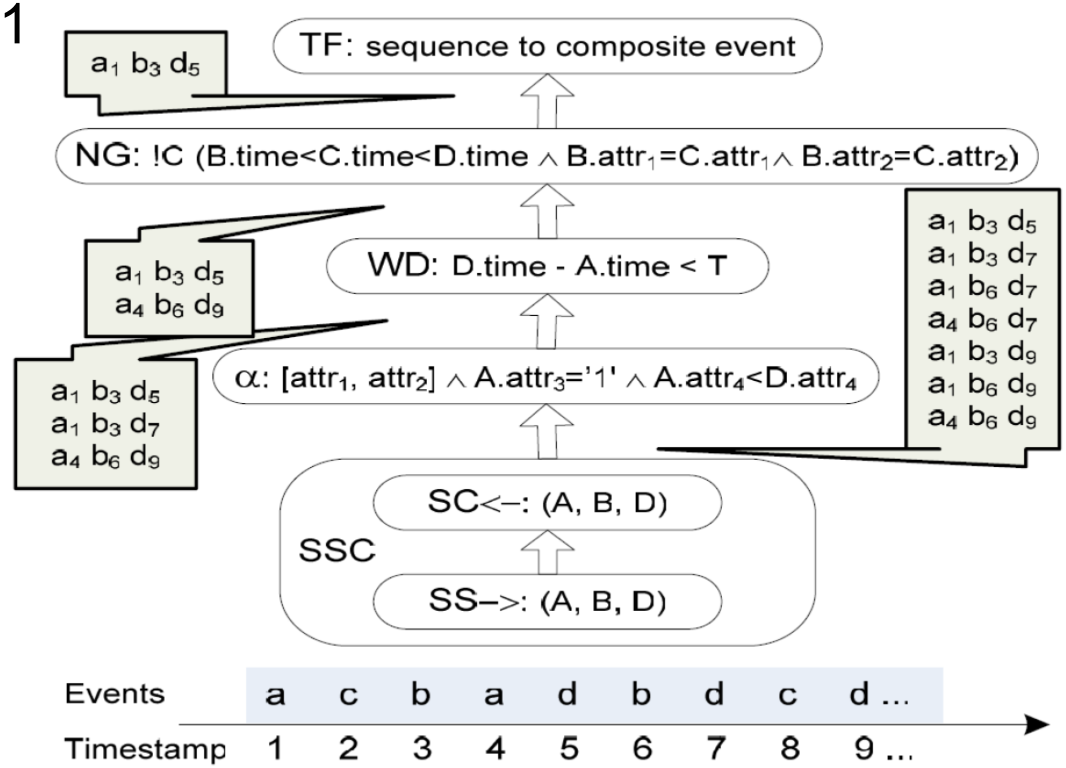
N.P. Schultz-Møller, M. Migliavacca, P.R. Pietzuch: *Distributed complex event processing with query rewriting*. DEBS 2009

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 x_1,B x_2,!C x_3,D x_4)
WHERE [a_1, a_2] AND $x_1.a_3$ = 1
 AND $x_1.a_4$ < $x_4.a_4$
WITHIN T



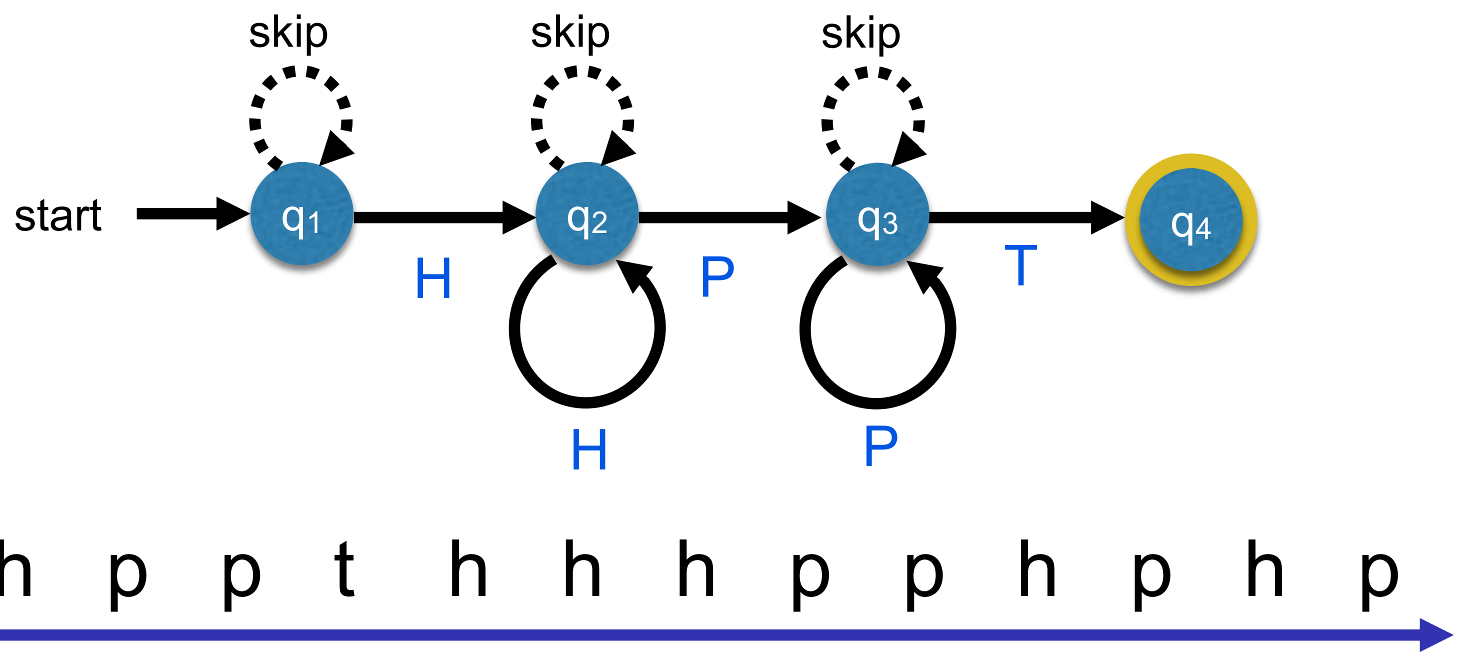
E. Wu, Y. Diao, S. Rizvi: High-performance complex event processing over streams. SIGMOD Conference 2006: 407-418

Issues of Automata-based Models 1/3

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)

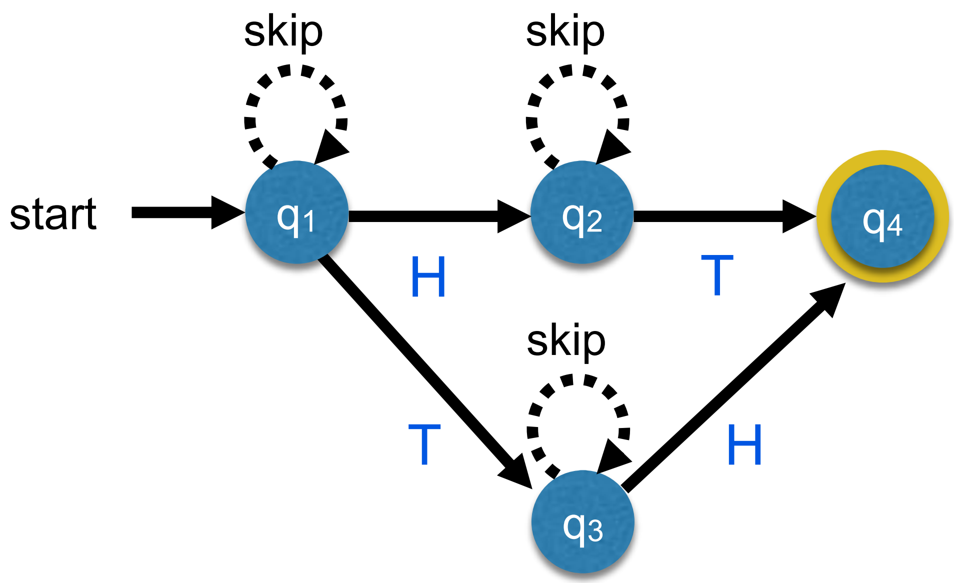


Issues of Automata-based Models 2/3

Forced sequential order

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

EVENT UNION(H h, T t)

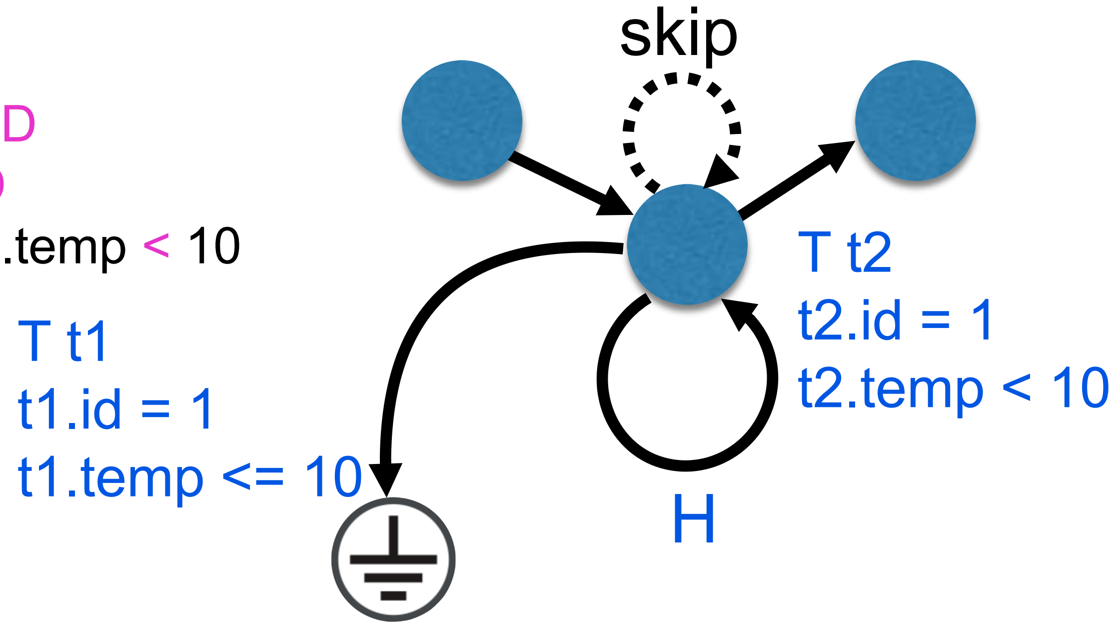


Issues of Automata-based Models 3/3

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
```

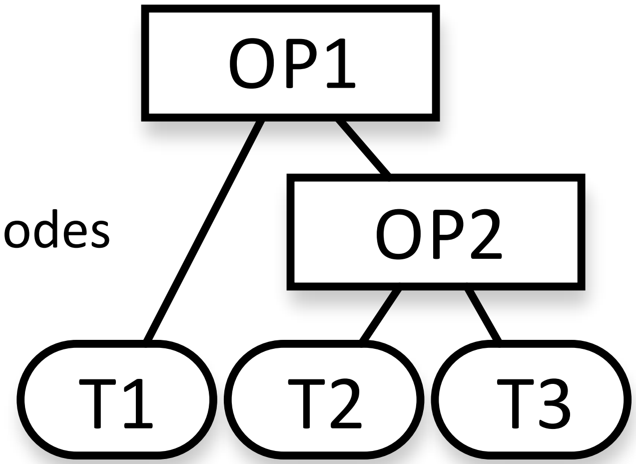


How to use tree-based models for event pattern specification?

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)



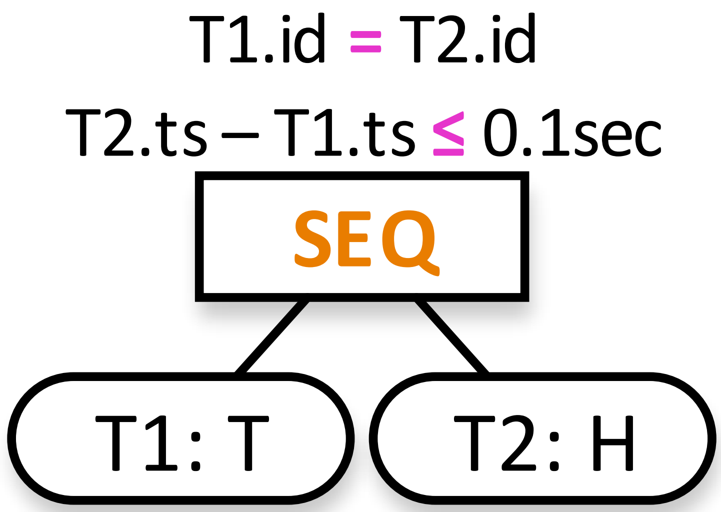
Different representations of a single pattern

– 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

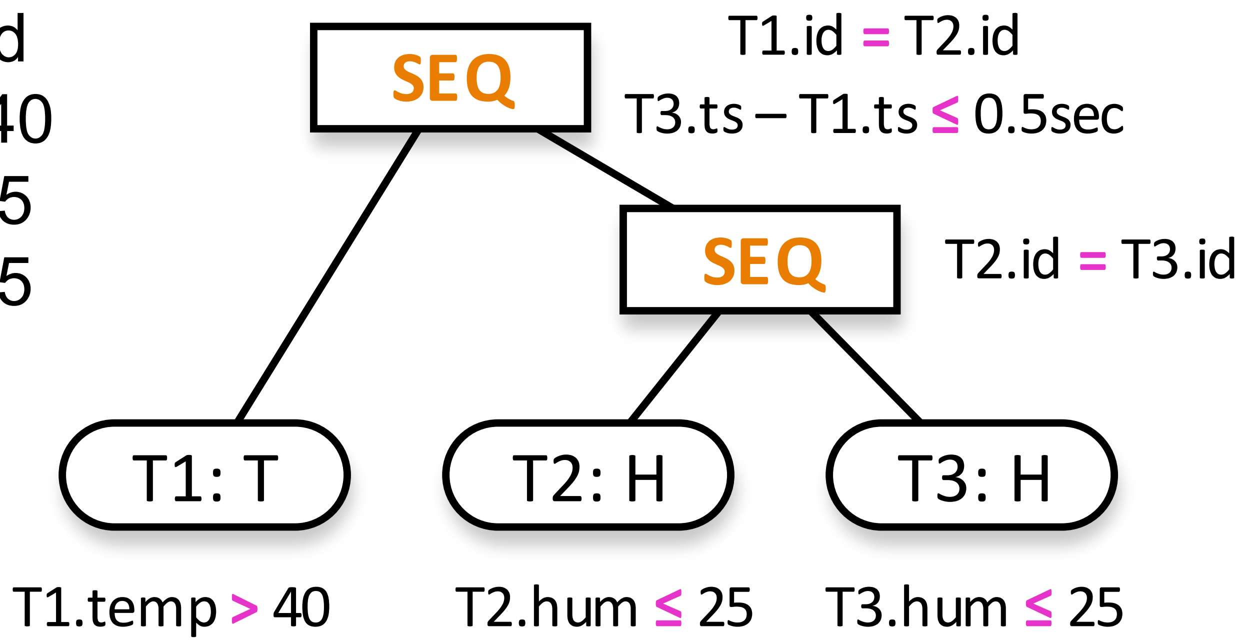
```
EVENT SEQ(T t, H h)
WHERE t.id = h.id
WITHIN 0.1
```



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

Extended Example Pattern

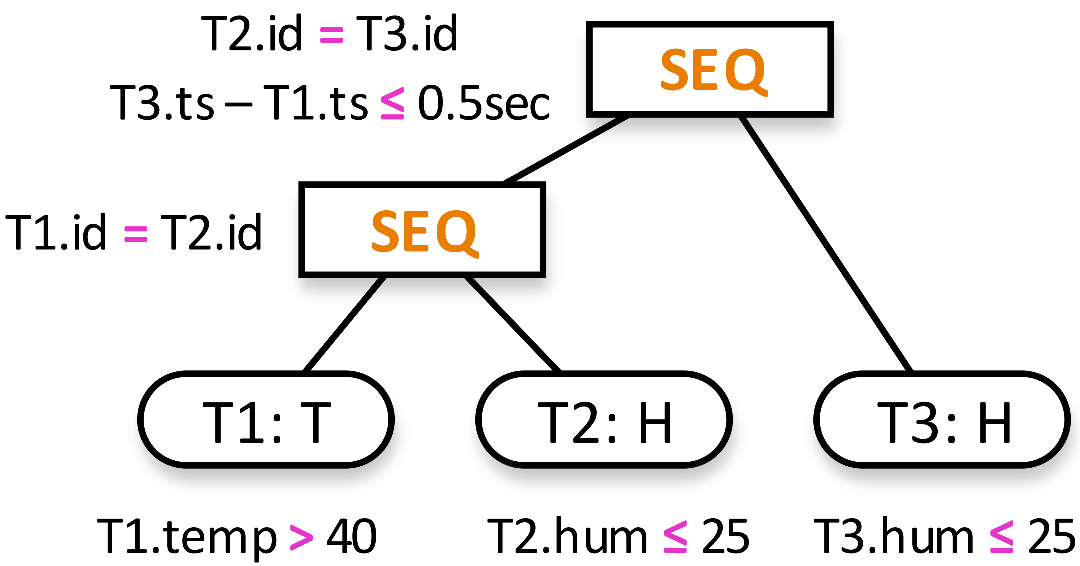
EVENT SEQ(T t, H h1, H h2)
WHERE t.id = h1.id
  AND h1.id = h2.id
  AND h1.temp > 40
  AND h2.hum ≤ 25
  AND h3.hum ≤ 25
WITHIN 0.5



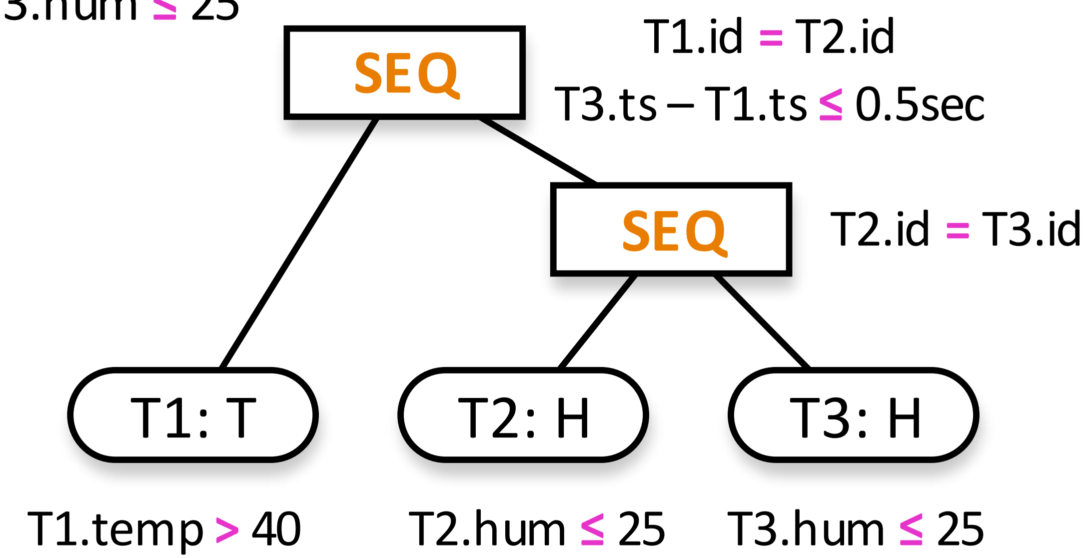
"All triples (`temperature`, `humidity`, `humidity`) with the same `id` such that high `temperature` is followed by multiple low `humidity` within 0.5 seconds"

Extended Example Pattern Again

Left-deep pattern

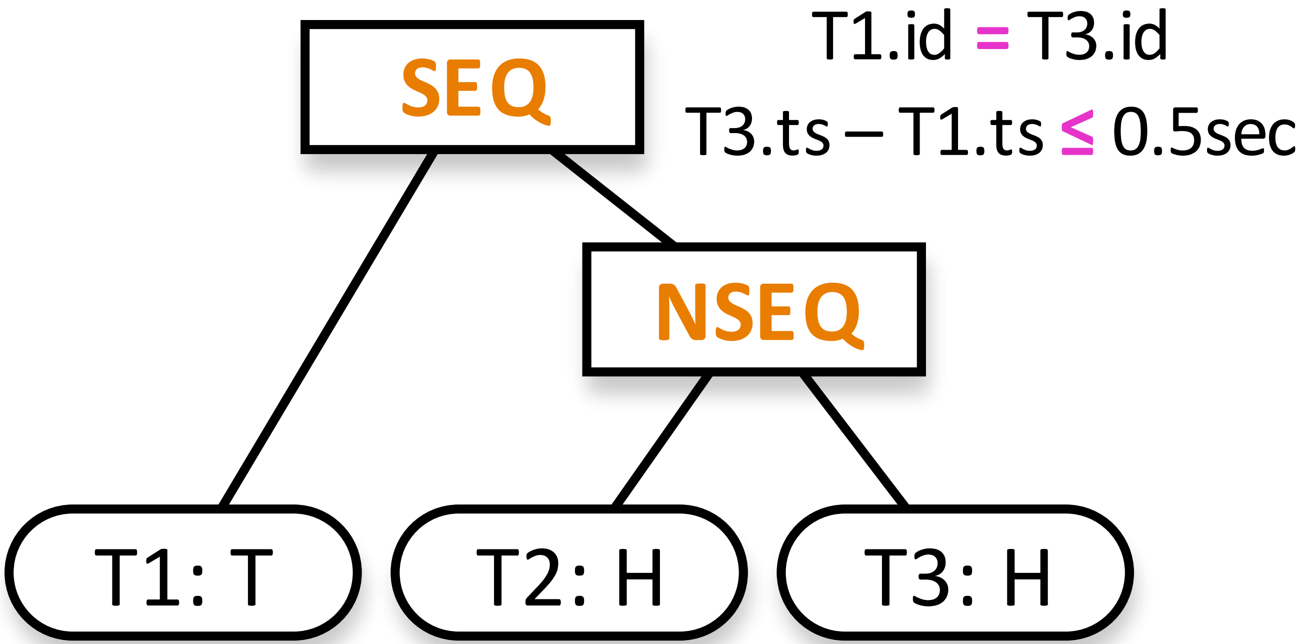


Right-deep pattern



Example Pattern with Negation

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

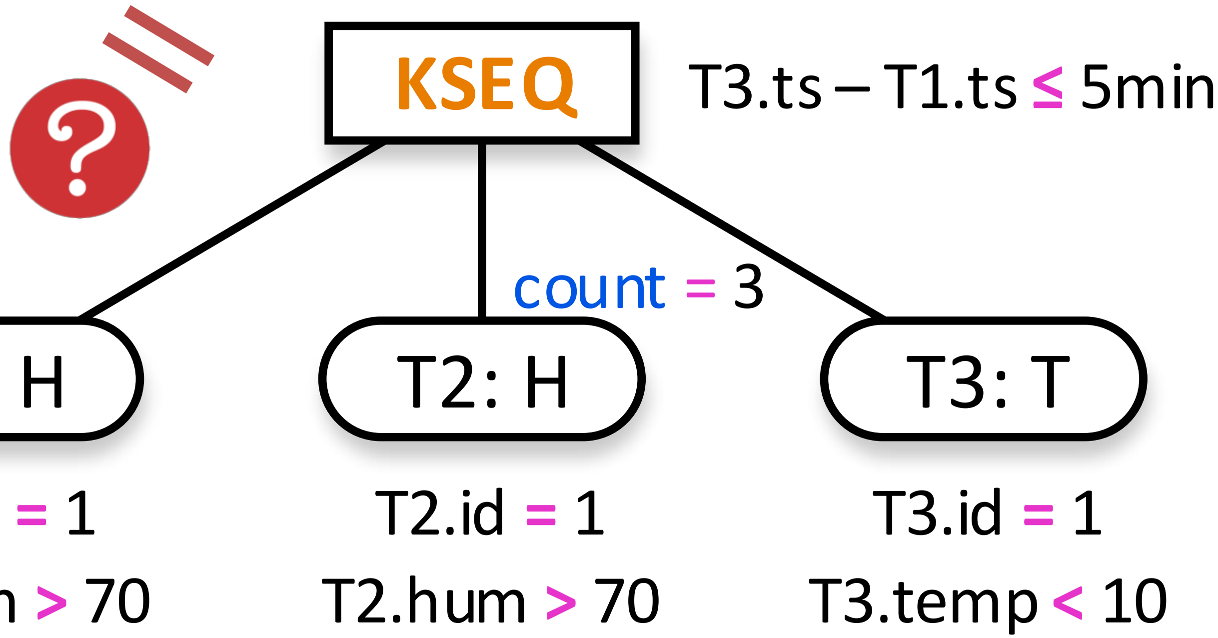


"All pairs (`temperature, humidity`), with the same `id` such that `humidity` is <u>the first</u> humidity measurement after `temperature`"

Example Pattern with Kleene Closure

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

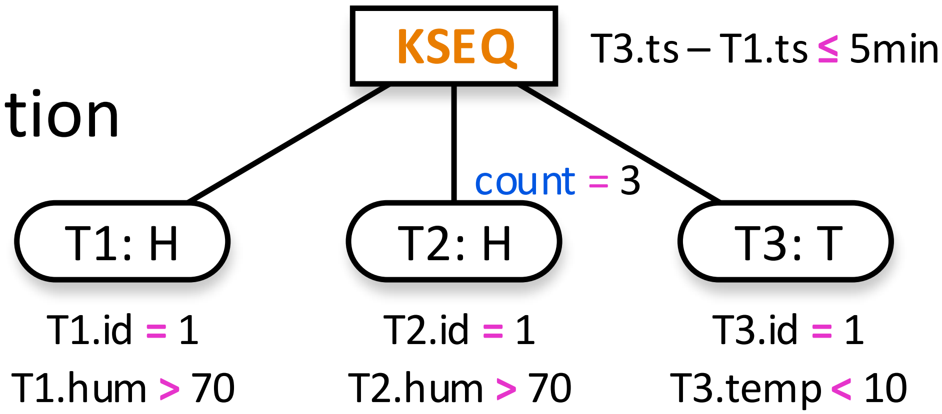
```
PATTERN SEQ(H+ h[], T t)
WHERE h.id = 1 AND t.id = 1
  AND h.hum > 70 AND t.temp < 10
WITHIN 5 minutes
HAVING count(h) = 4
```

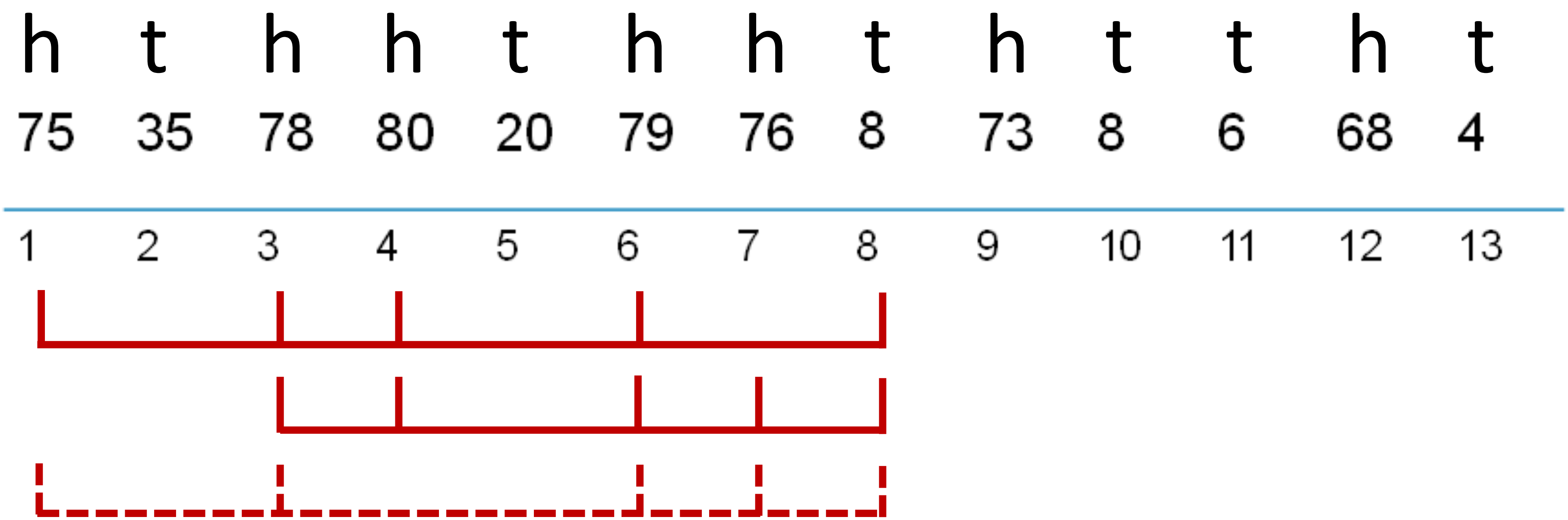


Semantics of KSEQ

KSEQ

- Semantics given by evaluation procedure
- In ZStream: Window of length count slides over buffer



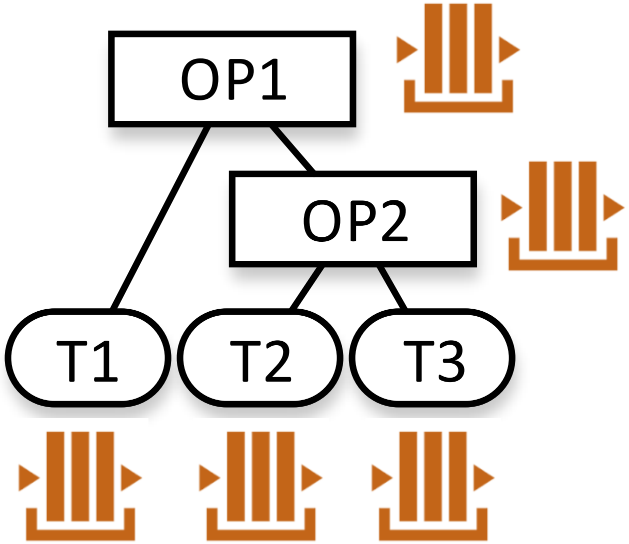


How to use tree-based models for event recognition?

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



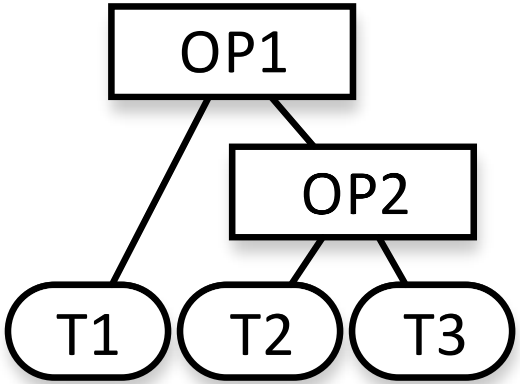
Major advantage

- 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

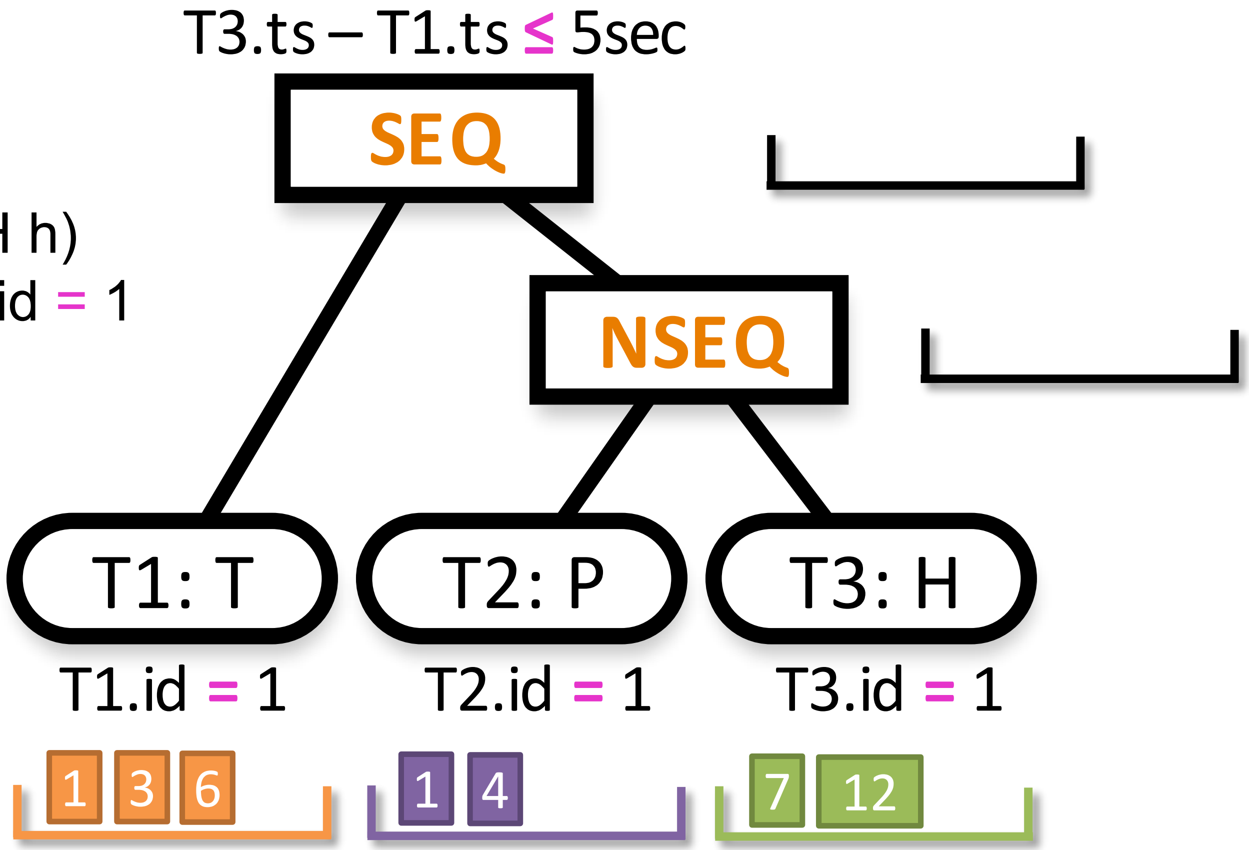


General procedure

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

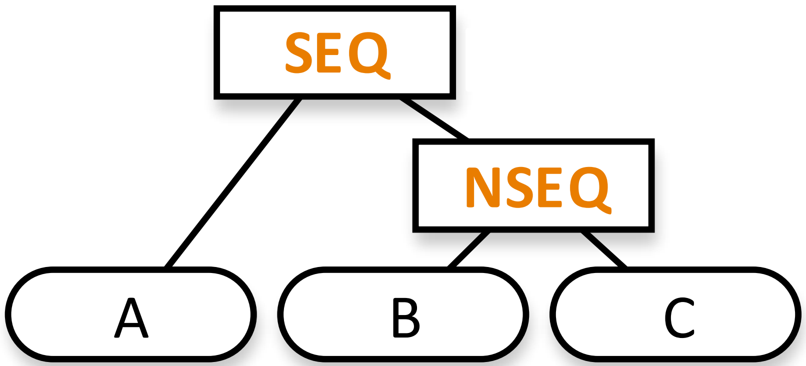


	p	t	t	p	t	p	t	t	h	h	t
id	1	1	1	1	2	3	1	2	1	1	1
ts	1	1	3	4	5	6	6	6	7	12	13

Evaluation of Negation Operator

Challenges:

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

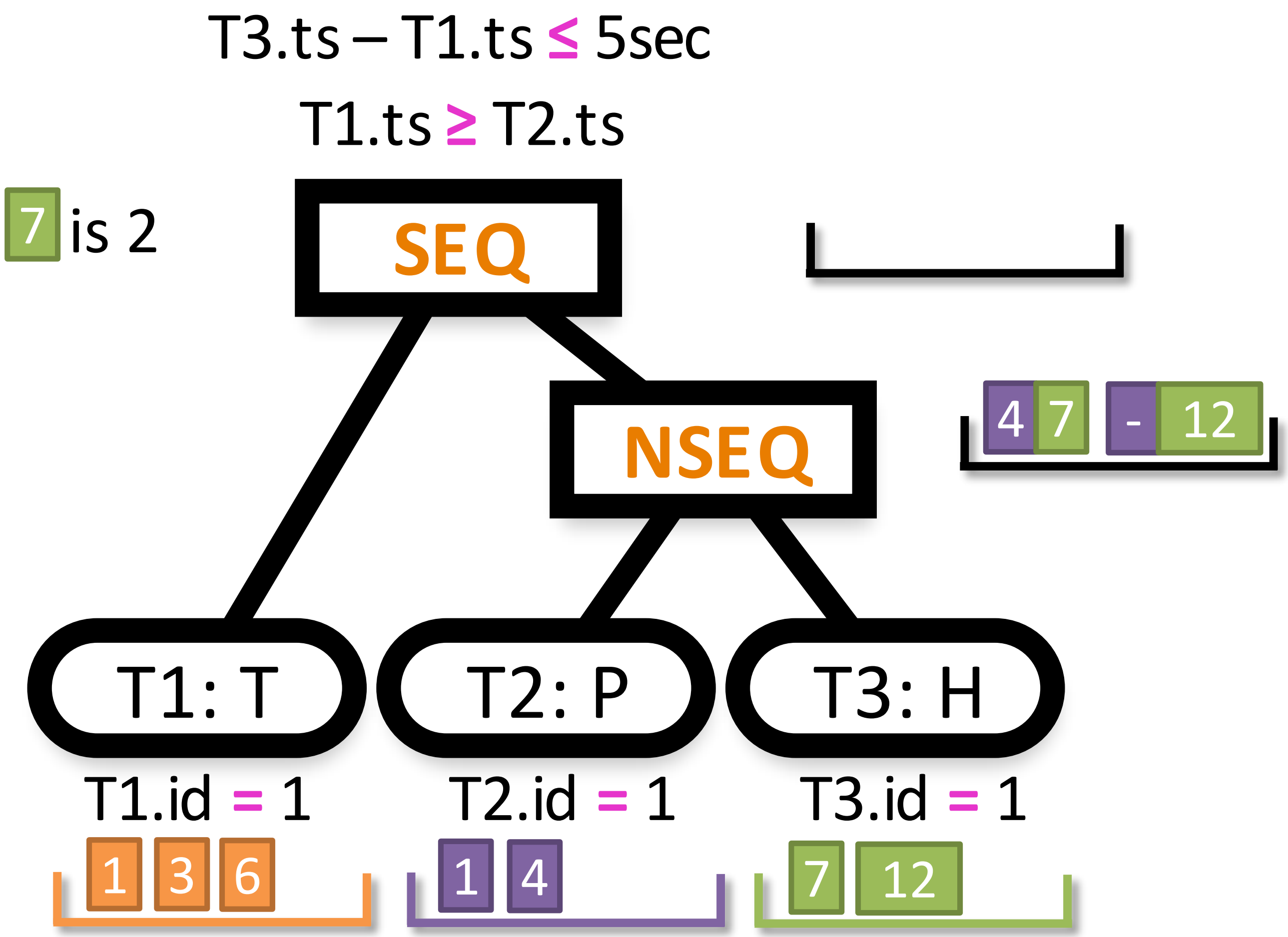


Approach:

- 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]



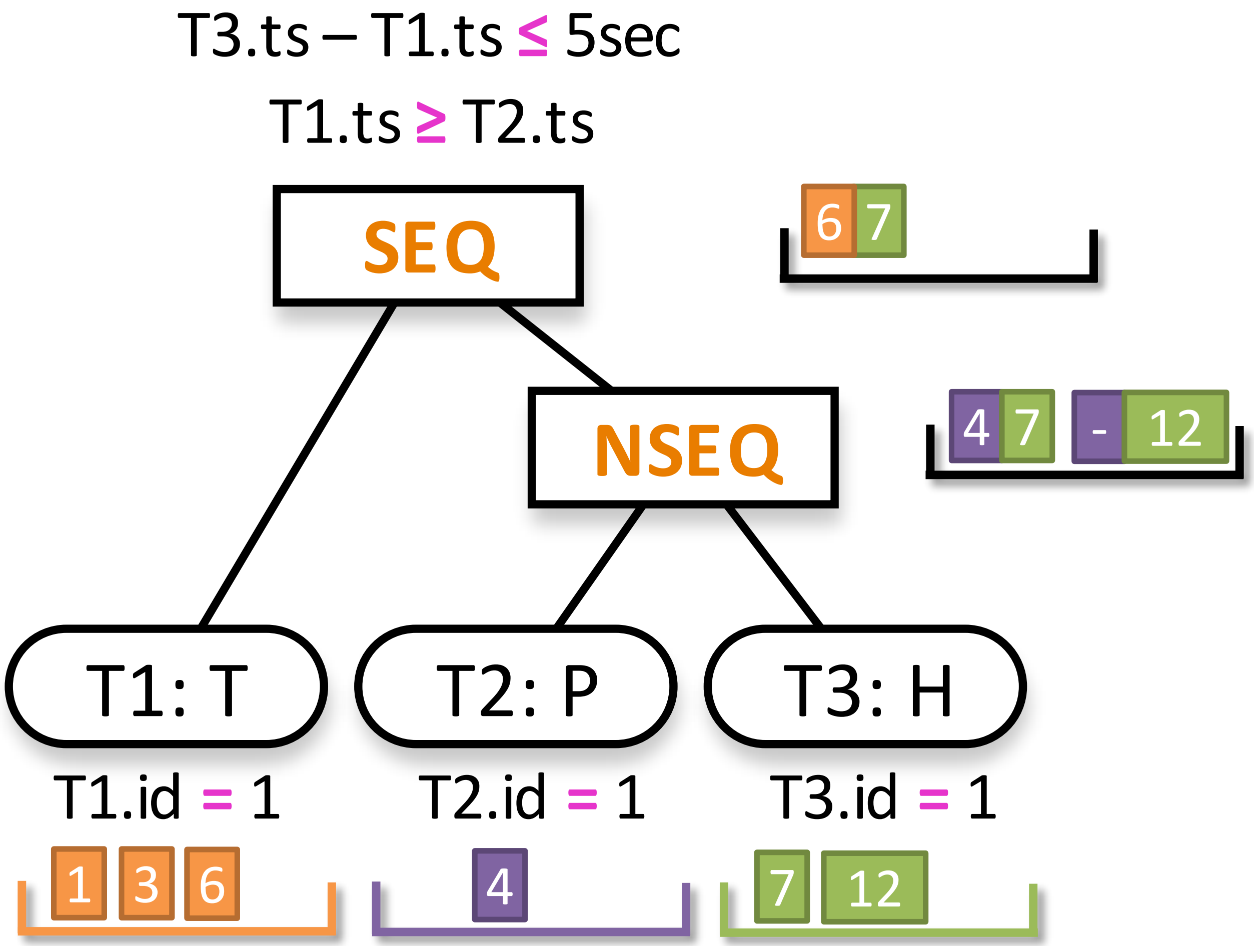
	p	t	t	p	t	p	t	t	h	h	t
id	1	1	1	1	2	3	1	2	1	1	1
ts	1	1	3	4	5	6	6	6	7	12	13

Evaluation of Sequence Operator

Iterate over right and left child buffer, filtering based on EAT

Check predicates for remaining events

1) EAT is 2
2) Filter 1
3) Assemble 6 7



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 x #input events touched`
- (Weighted) cost of assembling the output data:
  `#events created`

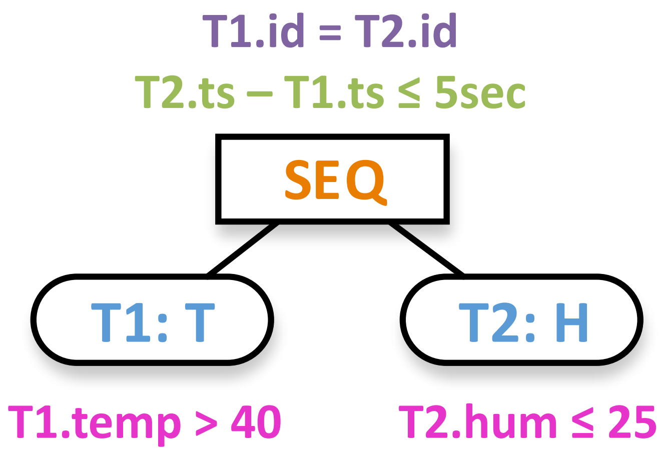
Cost of Sequence Operator

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

$C_X = rate_X * window * type\ selectivity$

$C_{T1} = 10e/sec * 5sec * 1/10 = 5e$

$C_{T2} = 20e/sec * 5sec * 1/5 = 20e$



Cost of accessing the input data:

$C_{In} = C_{T1} * C_{T2} * ordering\ selectivity$

$= 5e * 20e * 1/2 = 50e^2$

Cost of accessing the output data:

$C_{Out} = C_{In} * correlation\ selectivity$

$= 50e^2 * 1/5 = 10e^2$

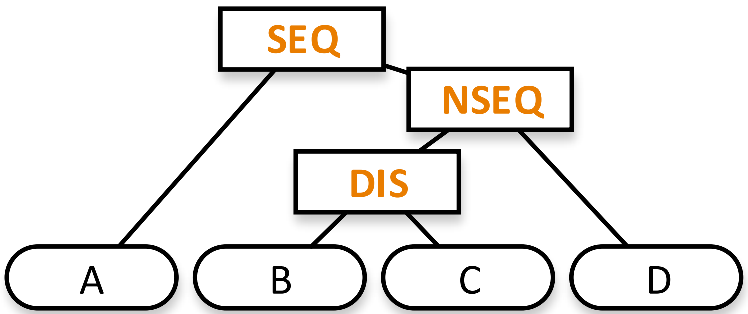
Plan Transformations

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)

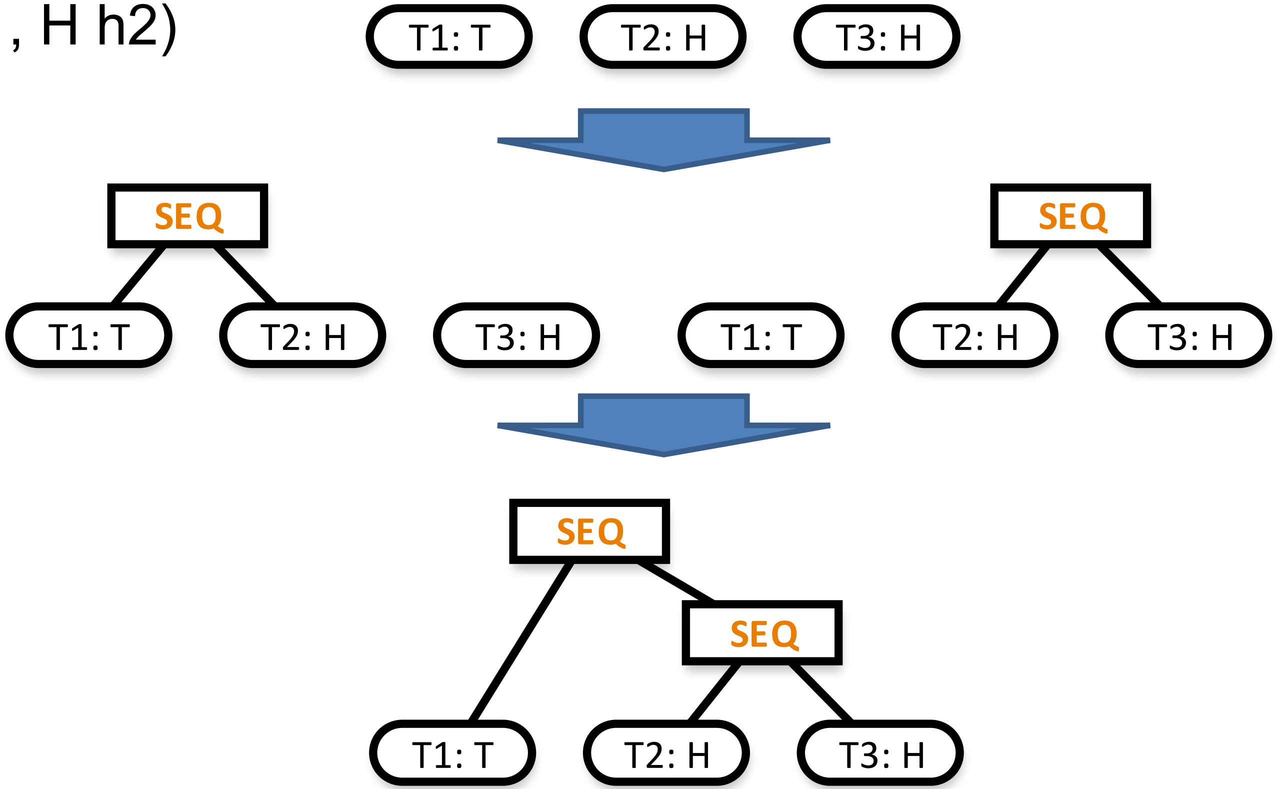


Plan Reordering

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

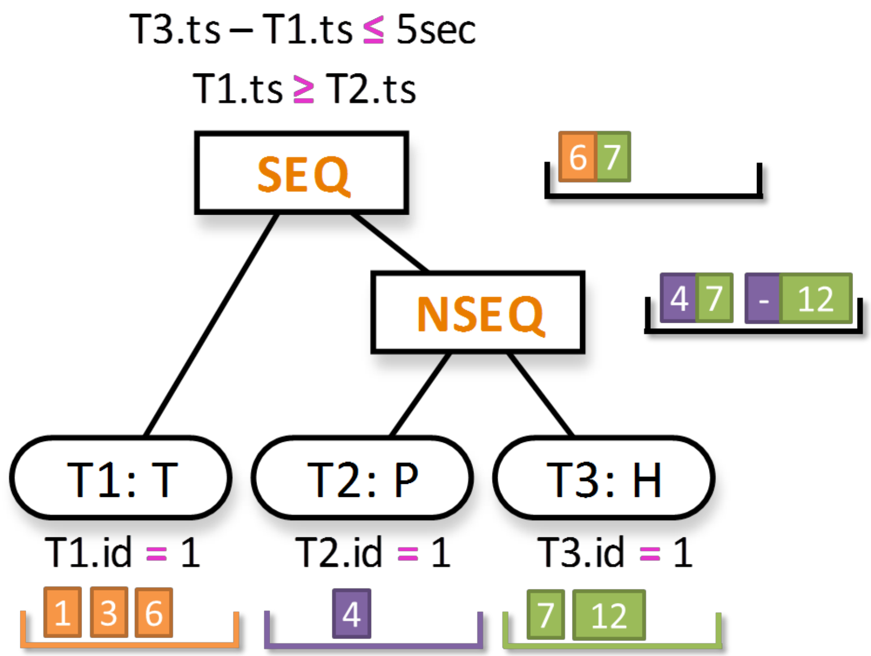
Trees of event operators…

…. 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:

A. Artikis, A. Skarlatidis, F. Portet and G. Paliouras. *Logic-based event recognition.* The Knowledge Engineering Review 2012

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

A. Artikis, M. Sergot and G. Paliouras. *An Event Calculus for Event Recognition*. TKDE 2014

D. Anicic, S. Rudolph, P. Fodor and N. Stojanovic. *Stream reasoning and complex event processing in ETALIS.* Semantic Web 2012

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

Predicate	Meaning
`event(E, T)`	Event `E` takes place at time `T`
`event(F:(?V1,?V2),T)`	An event takes place at time `T` changing the value of property `F` from `?V1` to `?V2`
`noevent(E, (T1,T2))`	Event `E` does not take place between [`T1,T2`)
`noevent(F:(?V1,?V2),(T1,T2))`	No event takes place between [`T1,T2`) that changes the value of property `F` from `?V1` to `?V2`
`hold(F:?V, (T1,T2))`	The value of property `F` is `?V` between [`T1,T2`)
`occurs(N,M,E,(T1,T2))`	Event `E` takes place at least `N` times and at most `M` times between [`T1,T2`)

Chronicle Representation Language

```
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

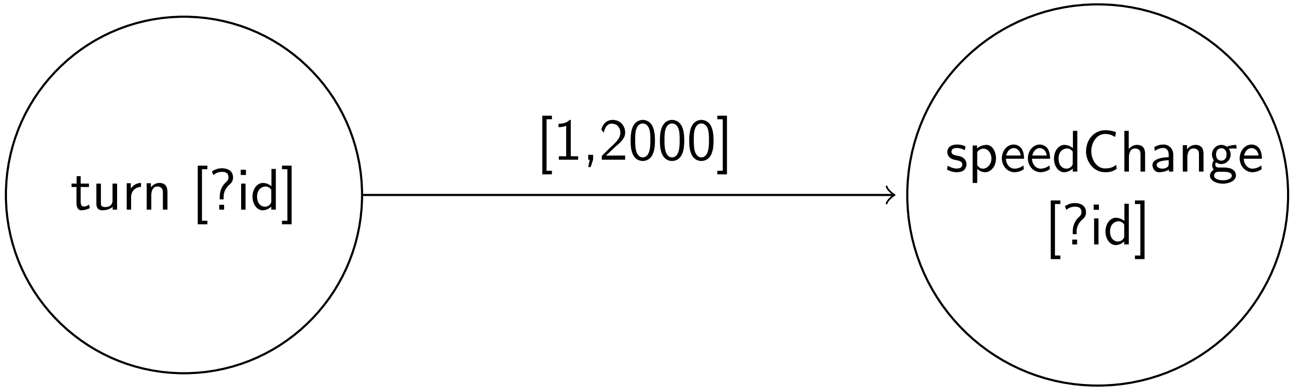
Chronicle Representation Language

- 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

Chronicle Recognition System: Semantics

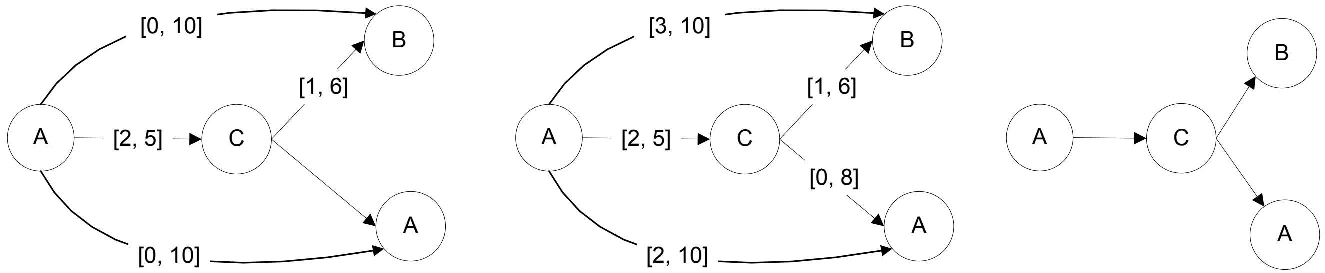
Pure temporal reasoning makes CRS efficient

Each CE definition is represented as a Temporal Constraint Network, as below



Chronicle Recognition System: Compilation

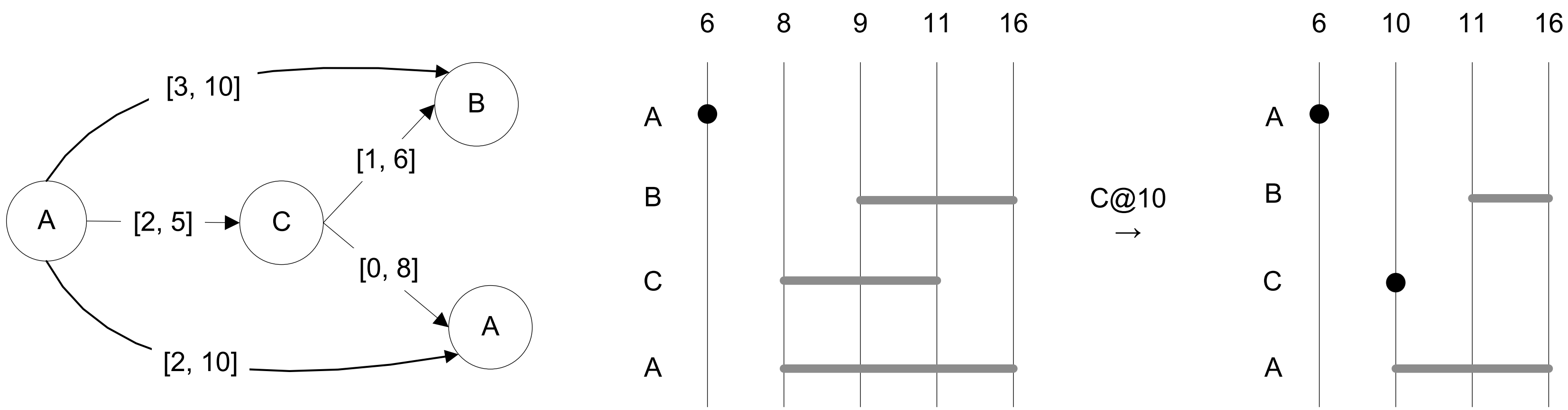
- Constraint propagation in the Temporal Constraint Network
  - Build the least constrained network
- Consistency checking
  - Detect inconsistencies at compile time



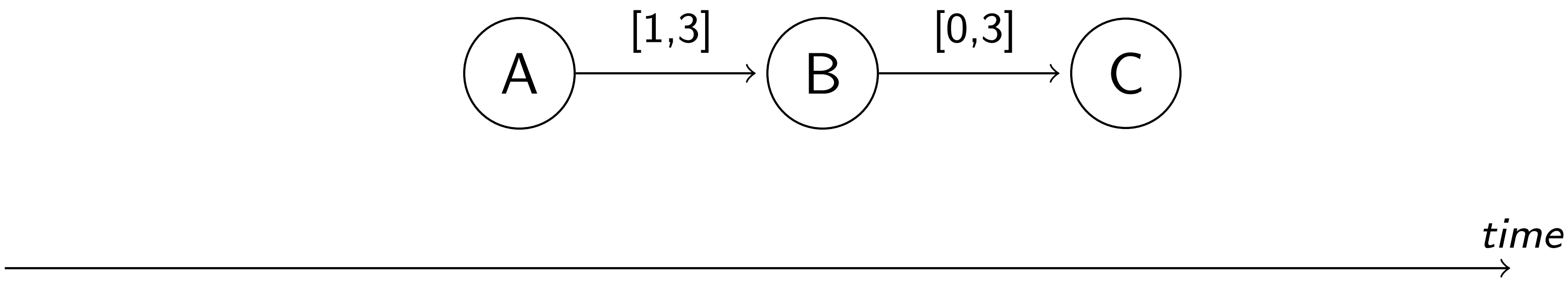
Chronicle Recognition System: Recognition

Recognition stage

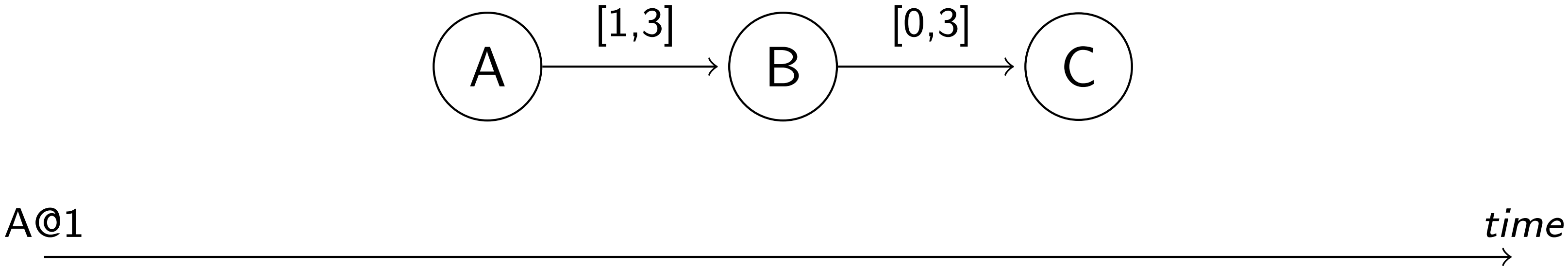
- Partial CE instance evolution
- Forward (incremental) recognition



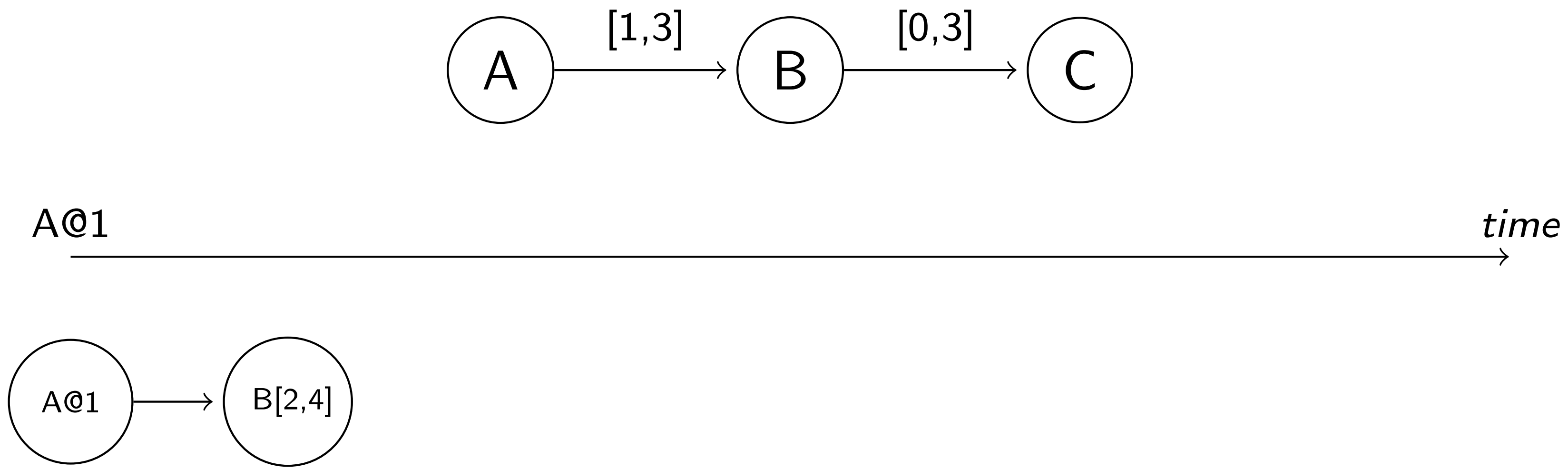
Chronicle Recognition System: Partial instances



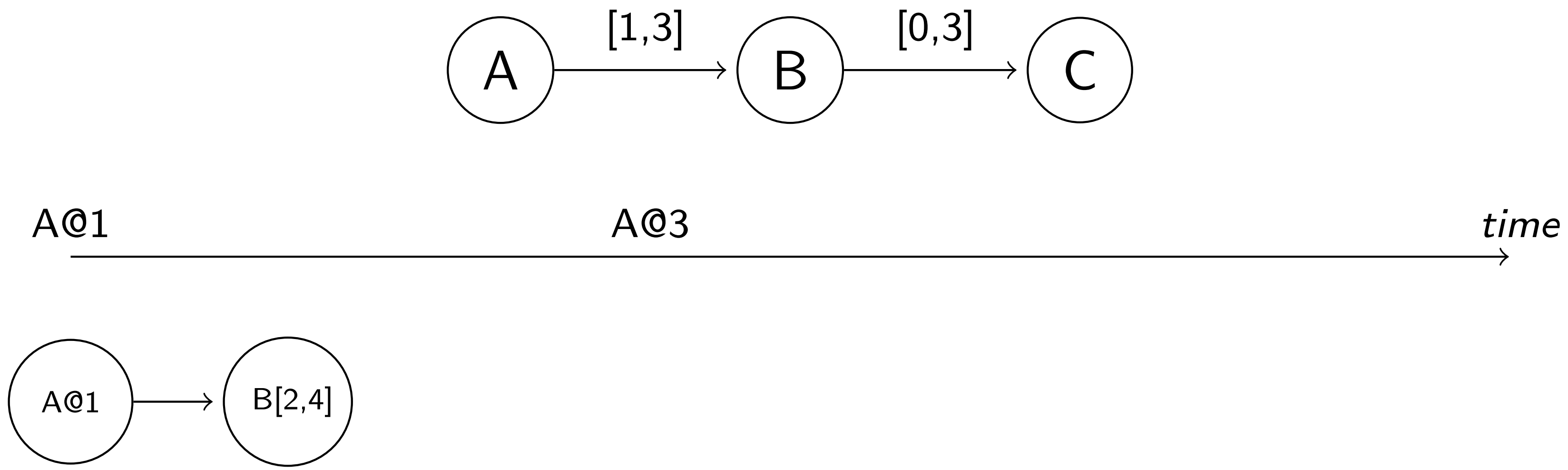
Chronicle Recognition System: Partial instances



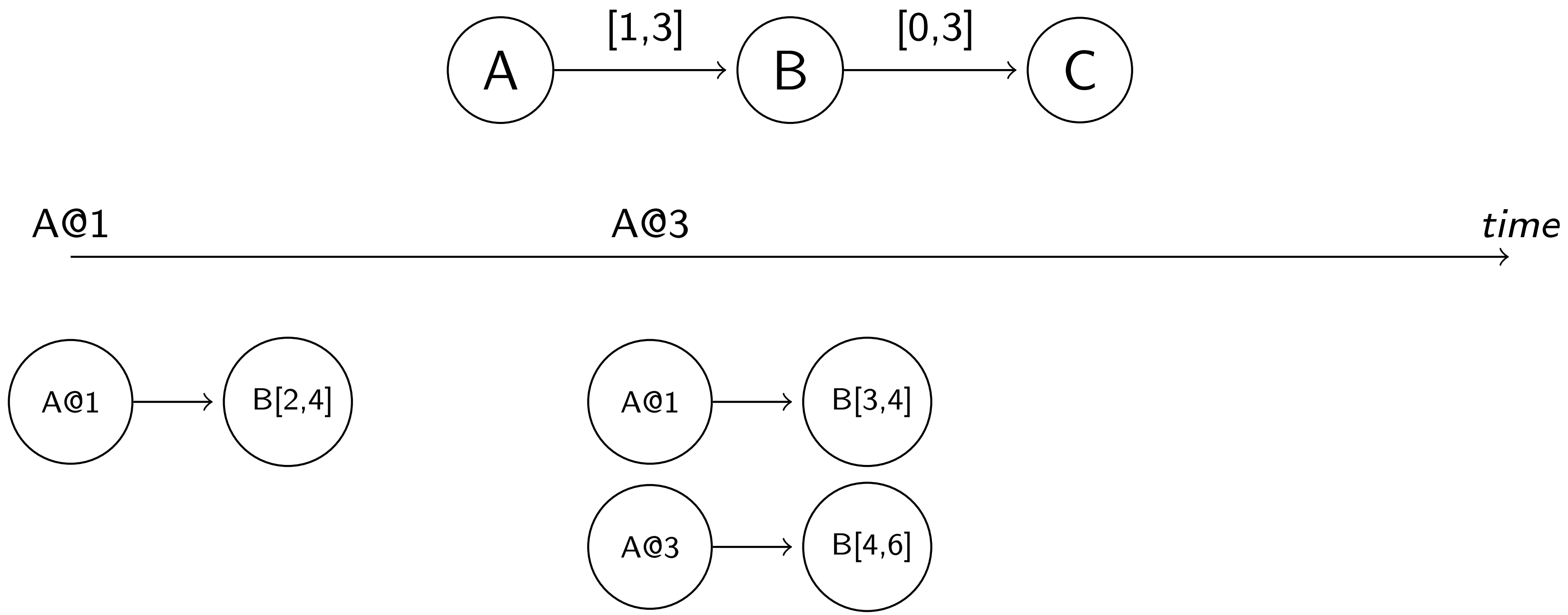
Chronicle Recognition System: Partial instances



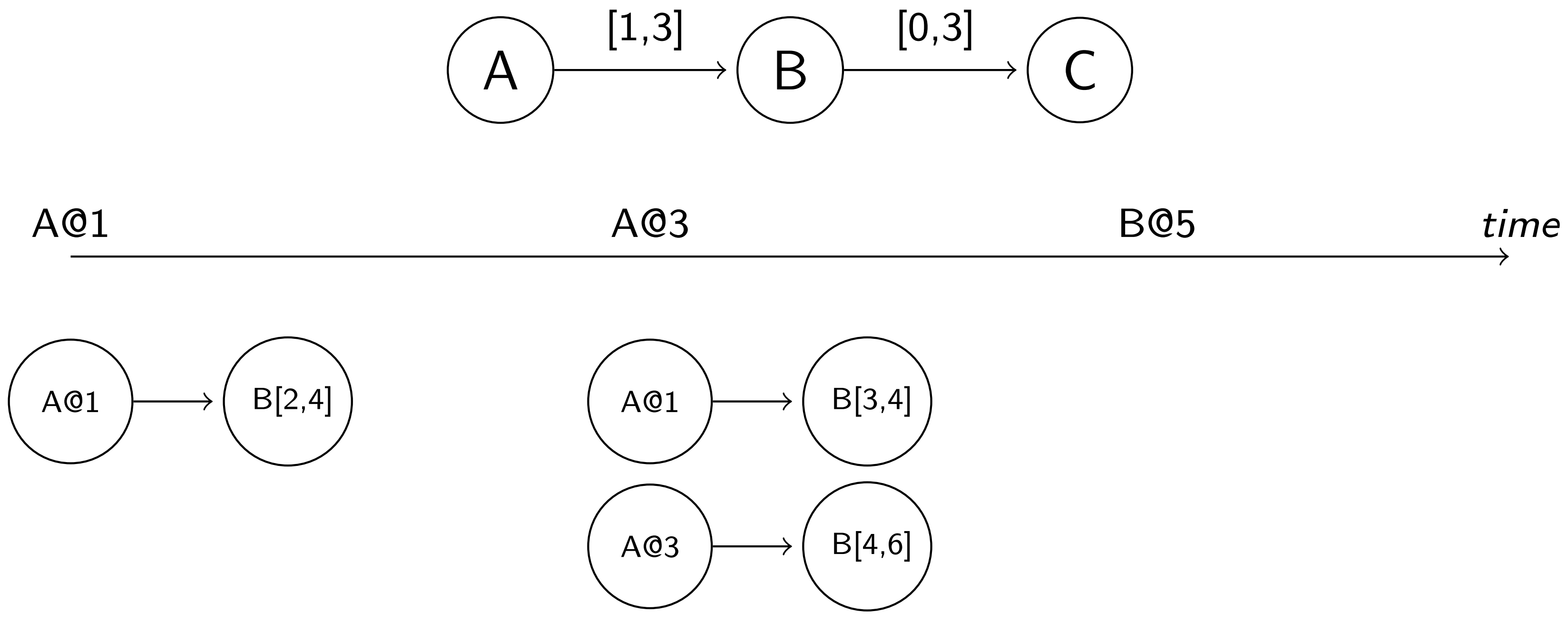
Chronicle Recognition System: Partial instances



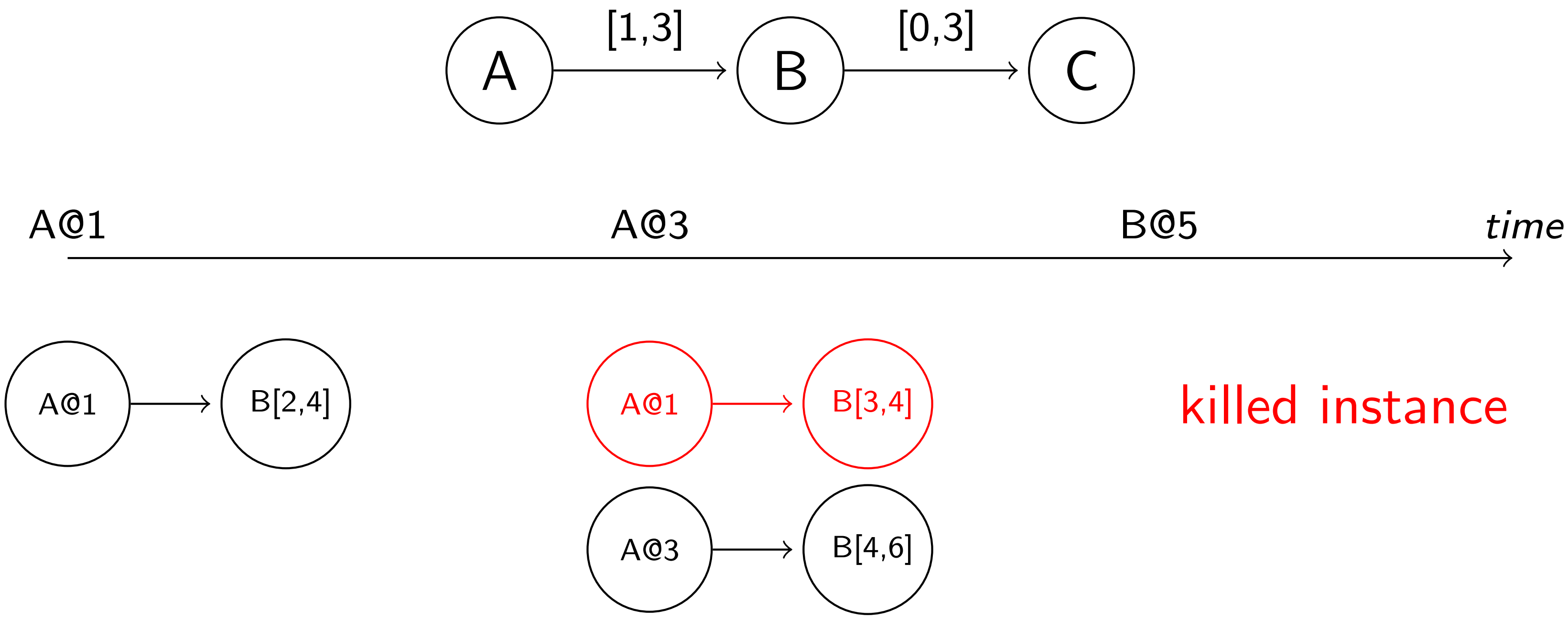
Chronicle Recognition System: Partial instances



Chronicle Recognition System: Partial instances

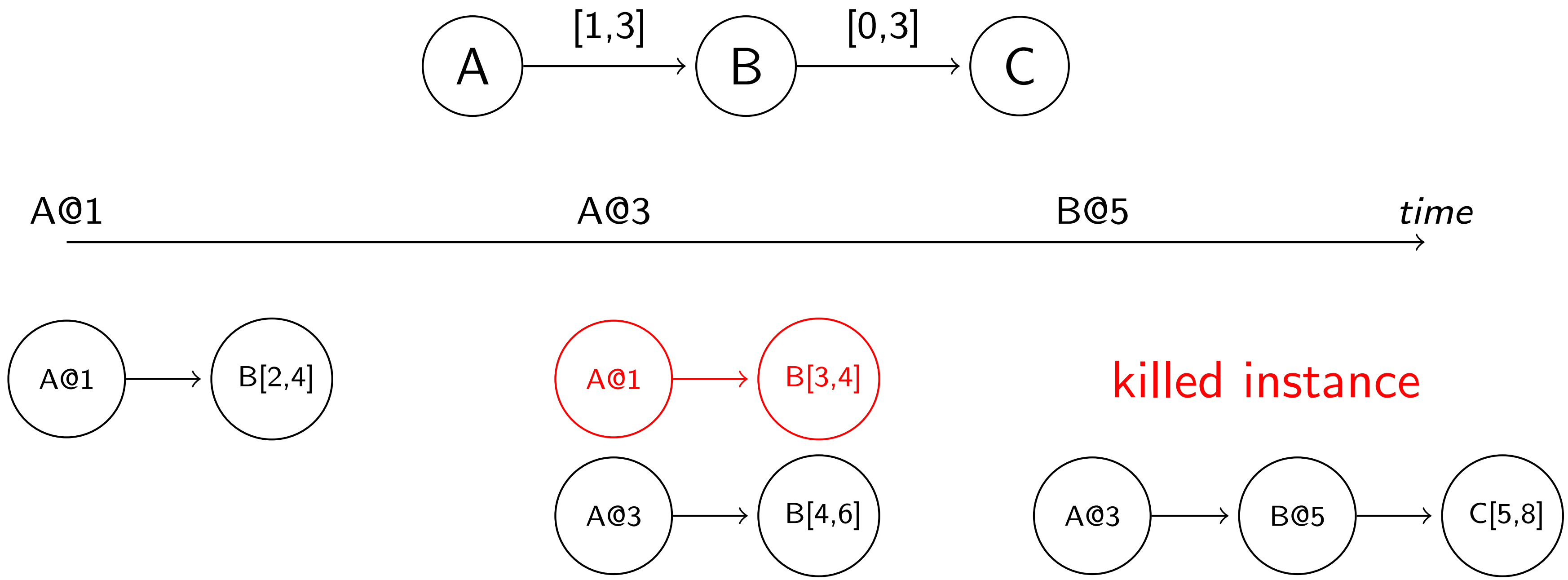


Chronicle Recognition System: Partial instances

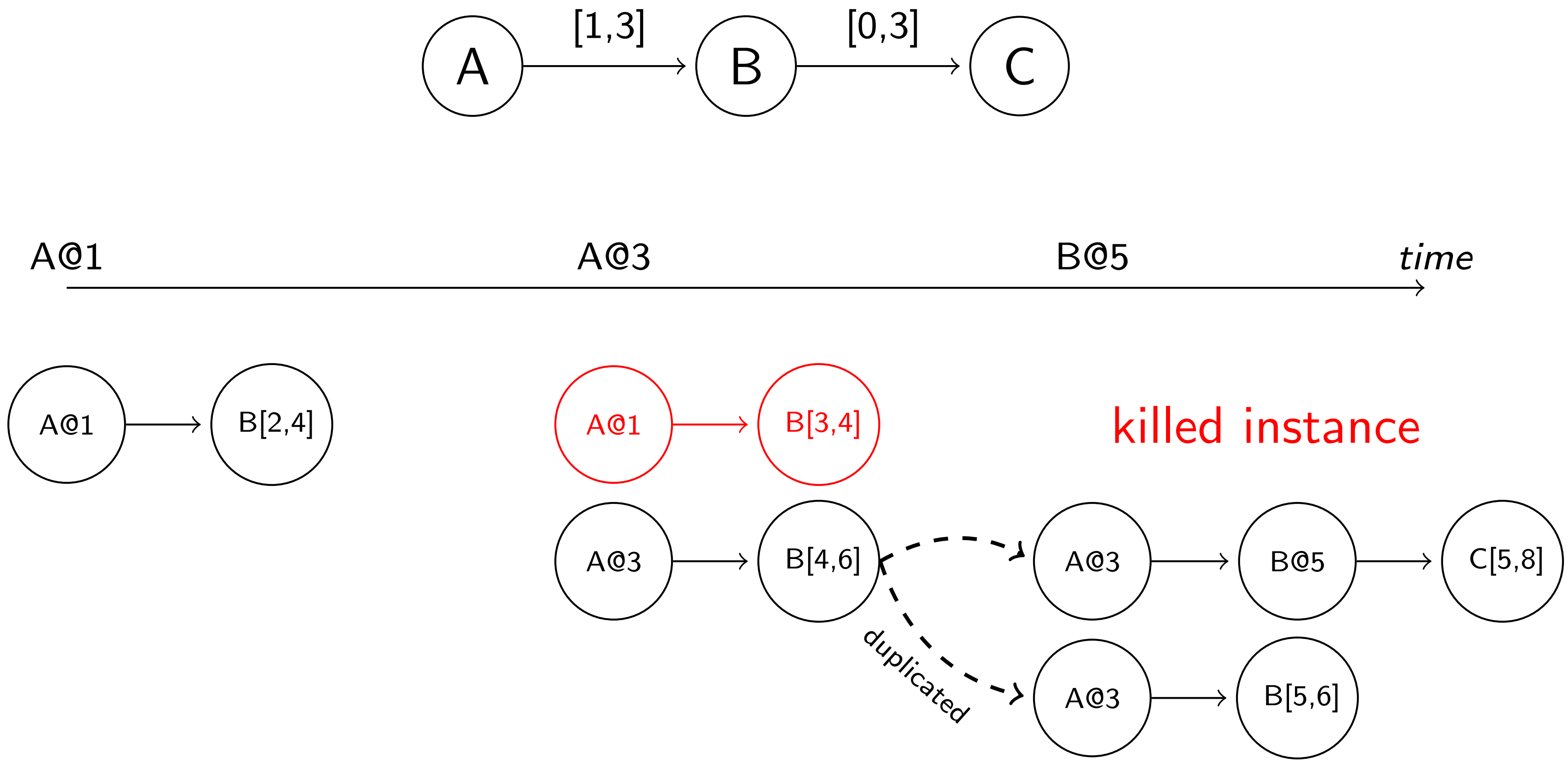


killed instance

Chronicle Recognition System: Partial instances



Chronicle Recognition System: Partial 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

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

TESLA

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

TESLA – Selection Policy

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

TESLA – Selection Policy

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

TESLA – Consumption Policy

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

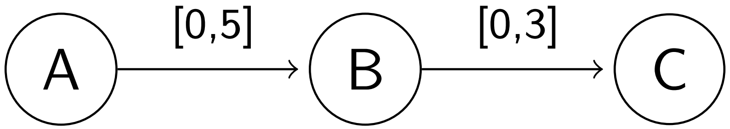
TESLA: Semantics

- 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

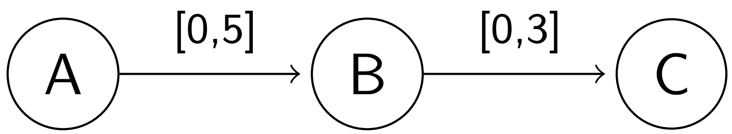
TESLA / T-Rex: Delayed Processing

- 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

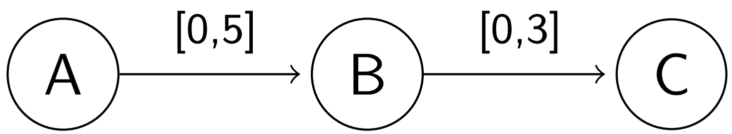


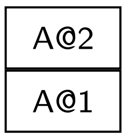
TESLA / T-Rex: Delayed Processing



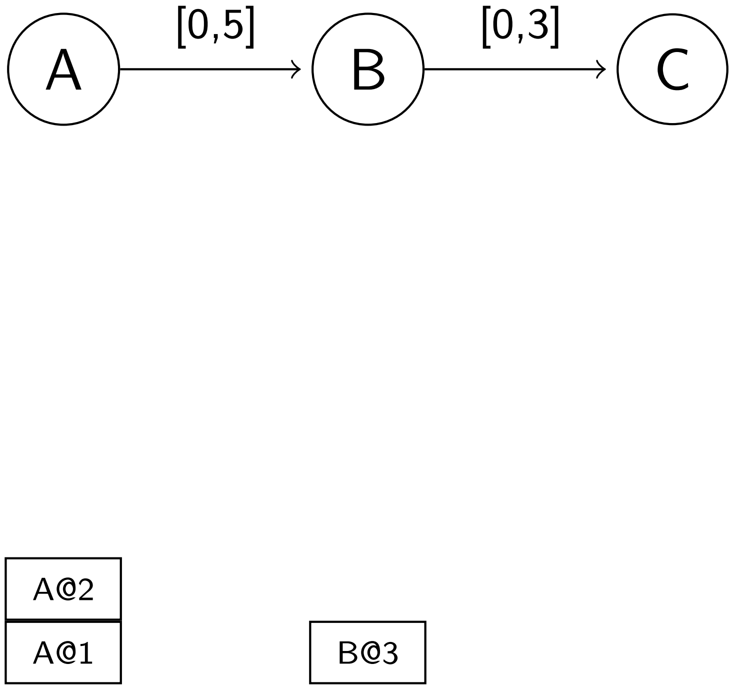
A@1

TESLA / T-Rex: Delayed Processing

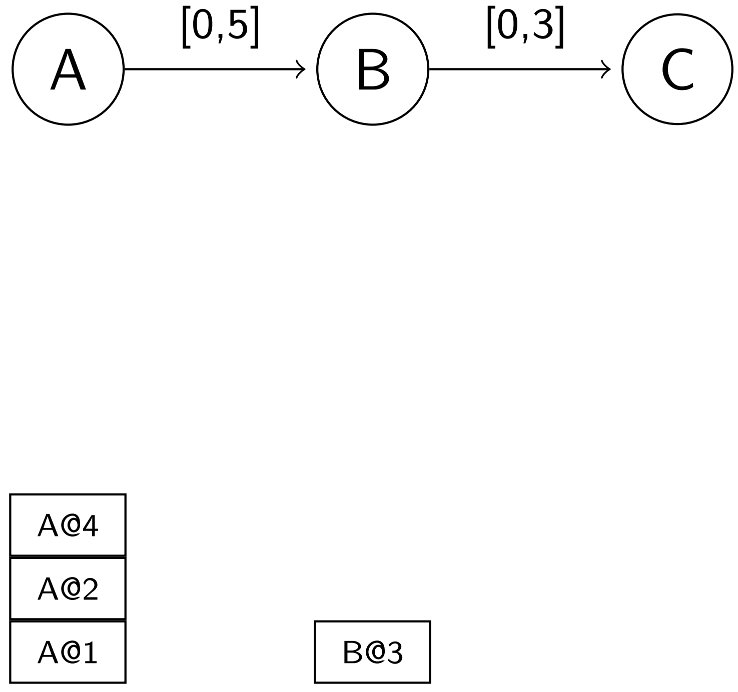




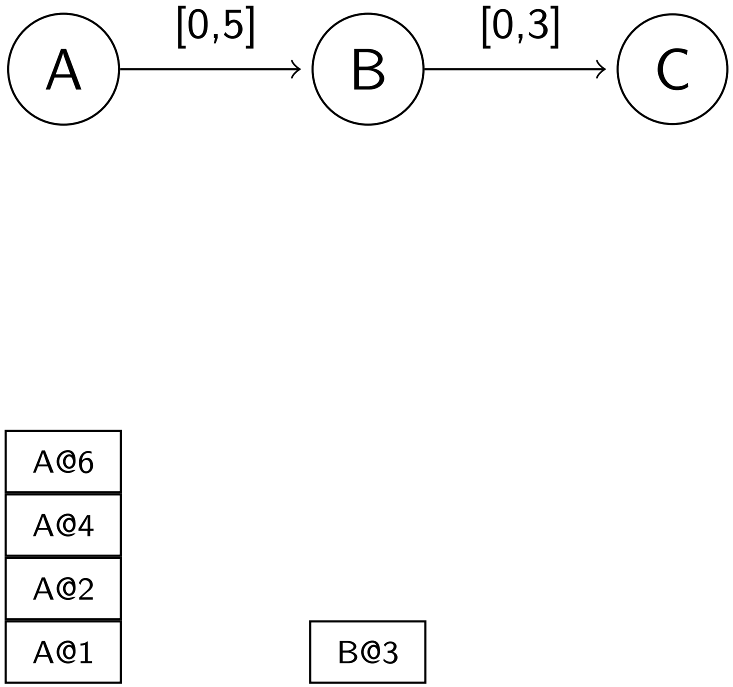
TESLA / T-Rex: Delayed Processing



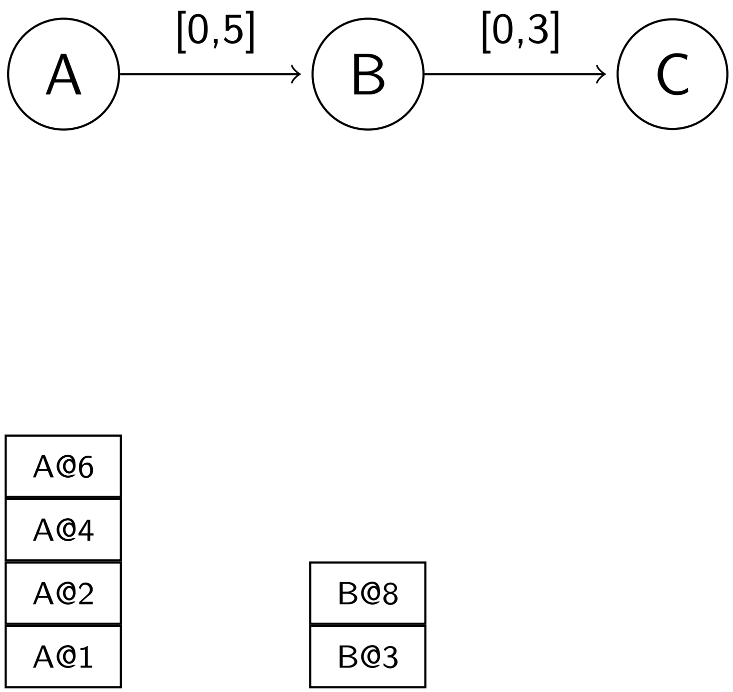
TESLA / T-Rex: Delayed Processing



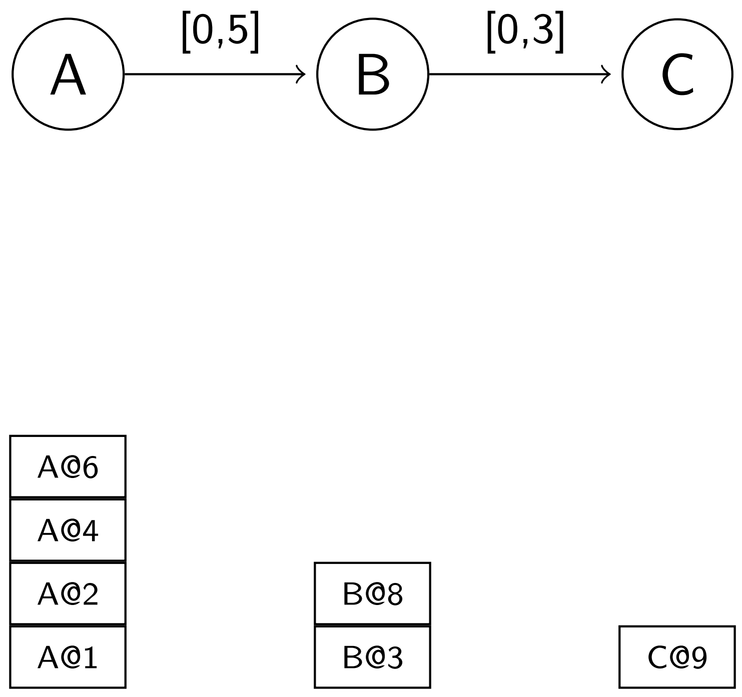
TESLA / T-Rex: Delayed Processing



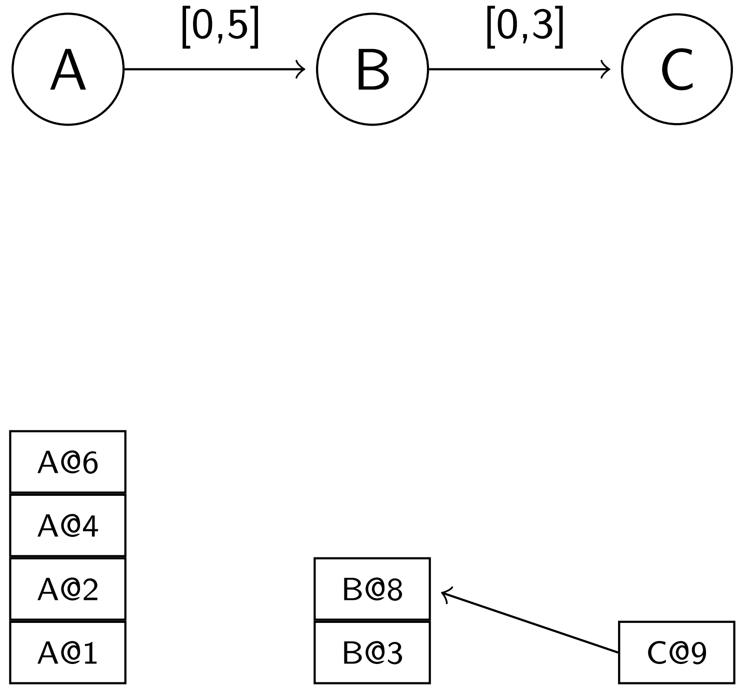
TESLA / T-Rex: Delayed Processing



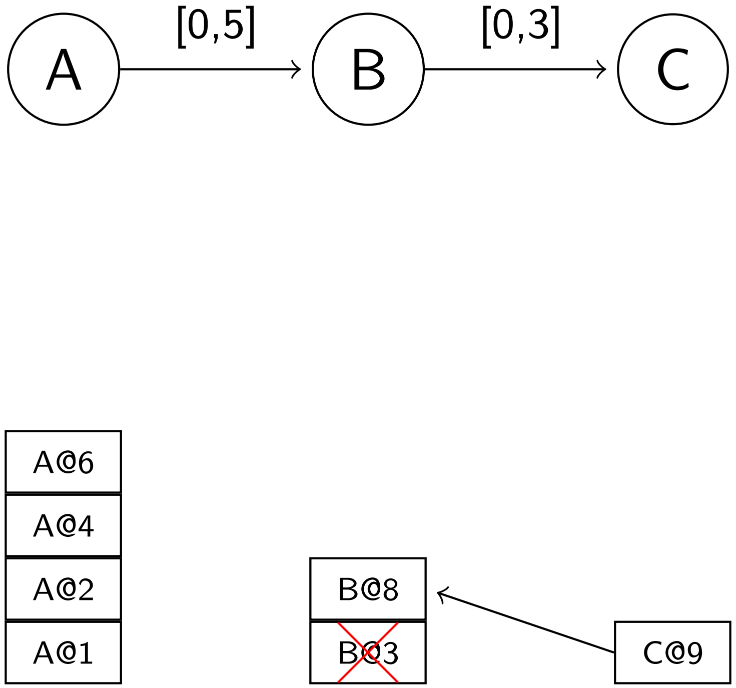
TESLA / T-Rex: Delayed Processing



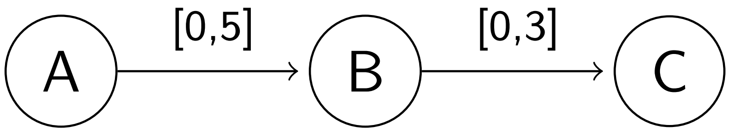
TESLA / T-Rex: Delayed Processing

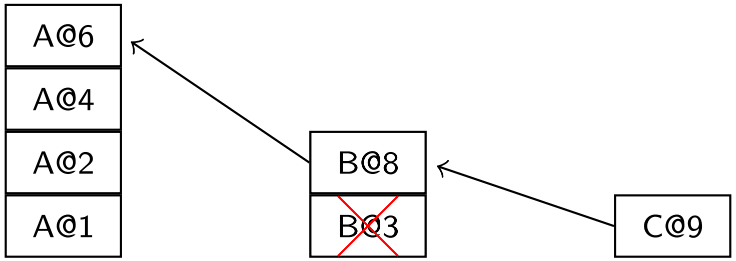


TESLA / T-Rex: Delayed Processing

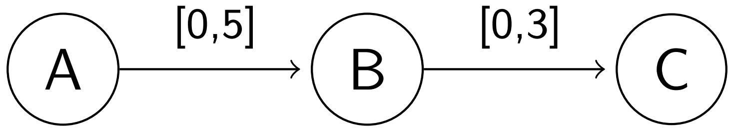


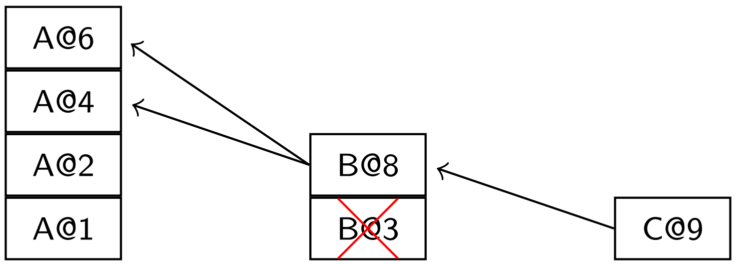
TESLA / T-Rex: Delayed Processing



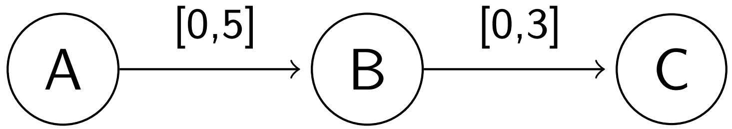


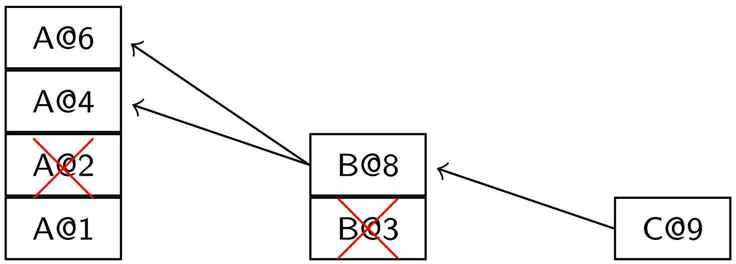
TESLA / T-Rex: Delayed Processing



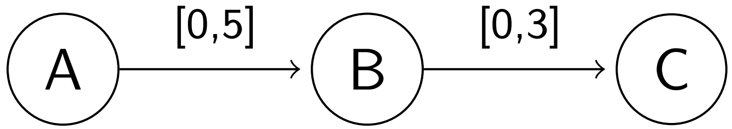


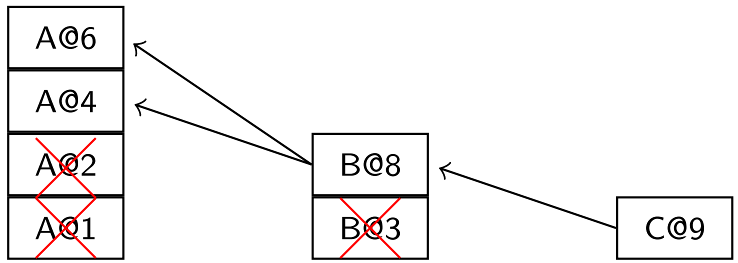
TESLA / T-Rex: Delayed Processing





TESLA / T-Rex: Delayed Processing





TESLA / T-Rex: Delayed Processing

- Avoid useless computations
- Avoid duplications
- Simple memory layout $\rightarrow$ Arrays
  - Cache friendly!

TESLA / T-Rex: Delayed Processing

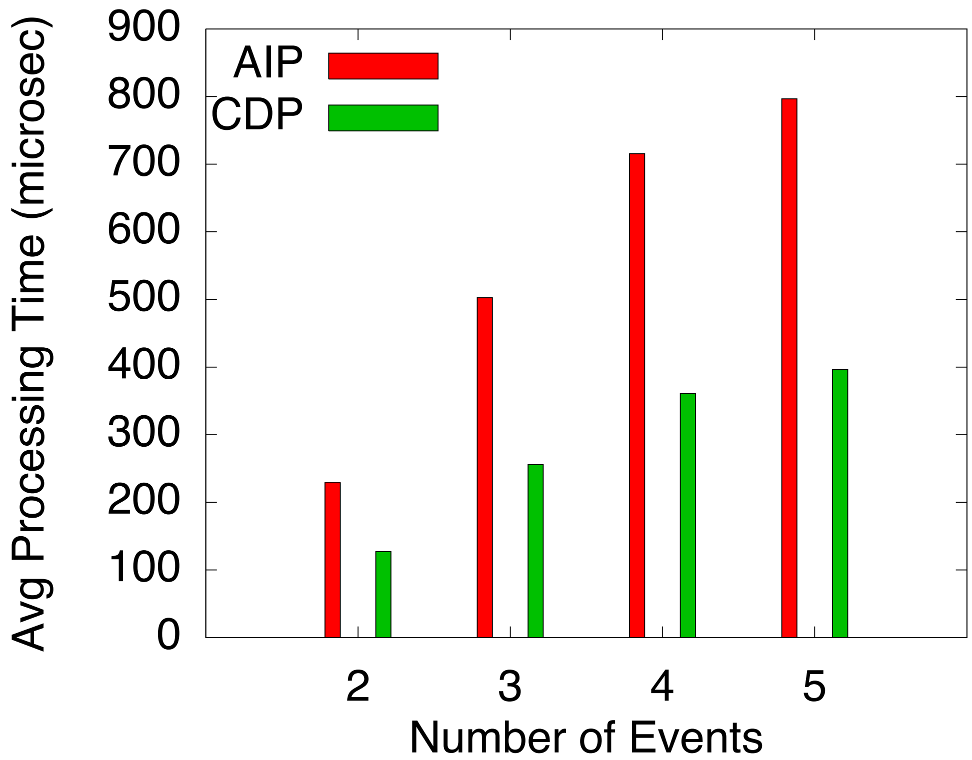


Figure: Each-within

TESLA / T-Rex: Delayed Processing

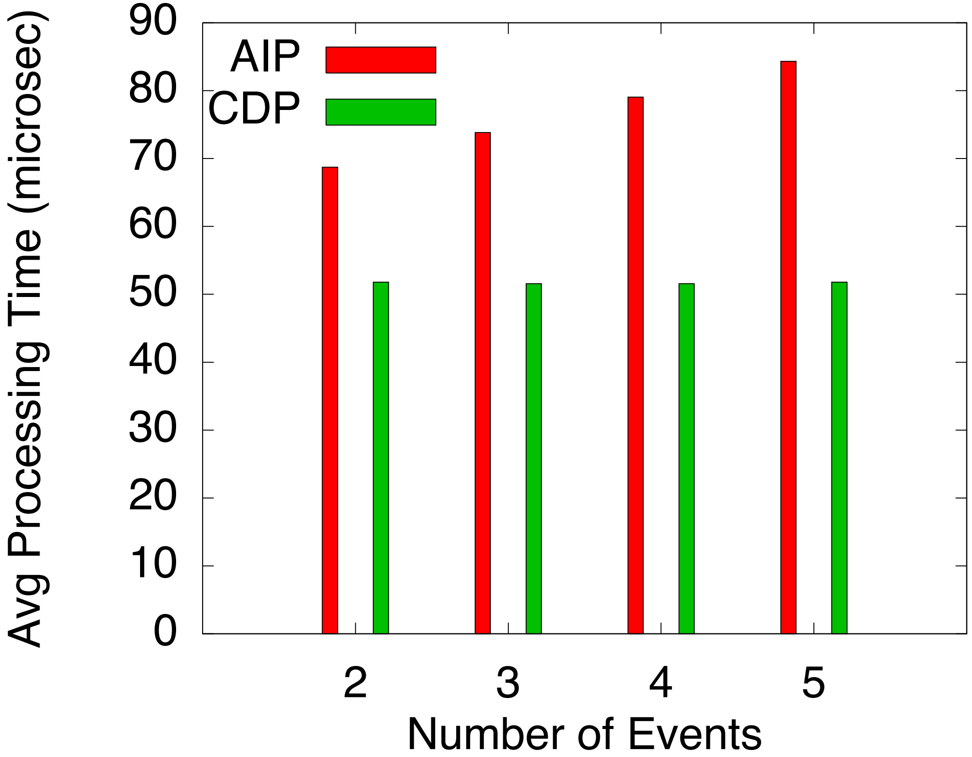
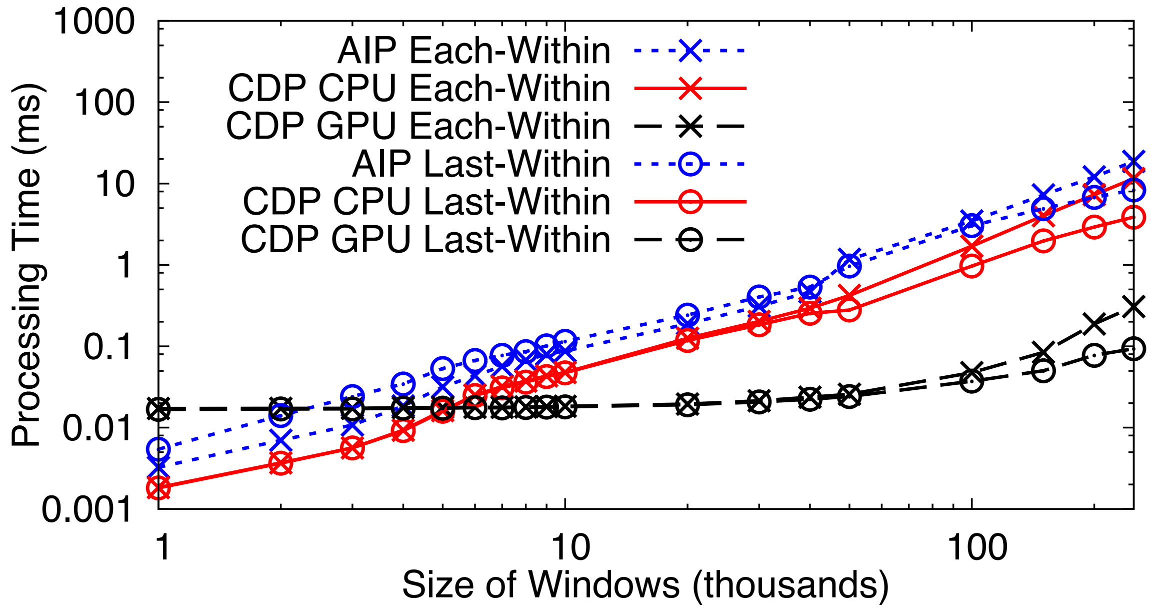


Figure: Last-within

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



TESLA / T-Rex: Summary

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

TESLA / T-Rex: Summary

- Formal semantics based on metric temporal logic formulas
- Flexible event selection and consumtion 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)

Predicate	Meaning
happensAt(E, T)	Event E occurs at time T
initiatedAt$(F = V, T)$	At time T a period of time for which $F = V$ is initiated
terminatedAt$(F = V, T)$	At time T a period of time for which $F = V$ is terminated
holdsFor$(F = V, I)$	I is the list of the maximal intervals for which $F = V$ holds continuously
holdsAt$(F = V, T)$	The value of fluent F is V at time T
union_all$([J_1, \ldots, J_n],\ I)$	$I = (J_1 \cup \ldots \cup J_n)$
intersect_all$([J_1, \ldots, J_n],\ I)$	$I = (J_1 \cap \ldots \cap J_n)$
relative_complement_all $(I',\ [J_1, \ldots, J_n],\ I)$	$I = I' \setminus (J_1 \cup \ldots \cup J_n)$

CE Definitions in RTEC: Simple Fluents

CE definition:

$$
\begin{array}{ll}
\textbf{initiatedAt}(CE,\ T) \leftarrow & \textbf{terminatedAt}(CE,\ T) \leftarrow \\
\quad \textbf{happensAt}(E_{In_1},\ T), & \quad \textbf{happensAt}(E_{T_1},\ T), \\
\quad [\text{conditions}] & \quad [\text{conditions}] \\
\ldots & \ldots \\
\textbf{initiatedAt}(CE,\ T) \leftarrow & \textbf{terminatedAt}(CE,\ T) \leftarrow \\
\quad \textbf{happensAt}(E_{In_i},\ T), & \quad \textbf{happensAt}(E_{T_j},\ T), \\
\quad [\text{conditions}] & \quad [\text{conditions}]
\end{array}
$$

where

$$
\text{conditions:} \qquad
\begin{array}{l}
{}^{0-K}\textbf{happensAt}(E_k,\ T), \\
{}^{0-M}\textbf{holdsAt}(F_m,\ T), \\
{}^{0-N}\text{atemporal-constraint}_n
\end{array}
$$

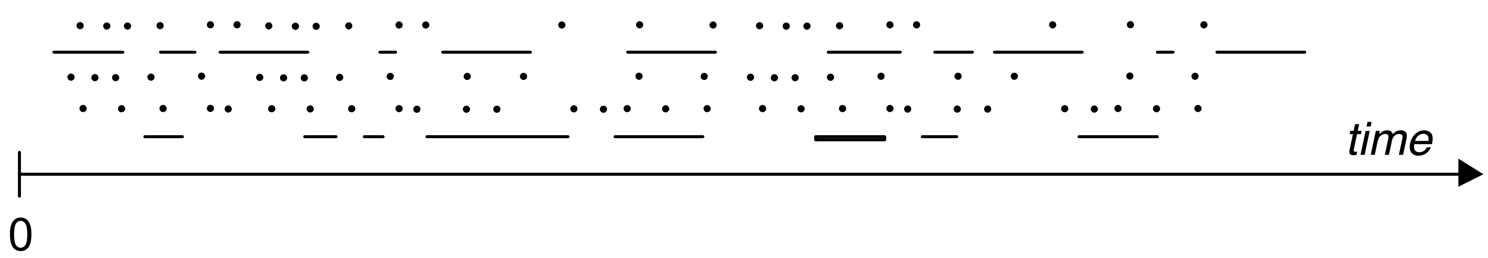
CE Definitions in RTEC: Simple Fluents

CE definition:

$$\begin{aligned}&\textbf{initiatedAt}(CE,\ T) \leftarrow\\ &\qquad \textbf{happensAt}(E_{In_1},\ T),\\ &\qquad [\text{conditions}]\\ &\ldots\\ &\textbf{initiatedAt}(CE,\ T) \leftarrow\\ &\qquad \textbf{happensAt}(E_{In_i},\ T),\\ &\qquad [\text{conditions}]\end{aligned}$$

$$\begin{aligned}&\textbf{terminatedAt}(CE,\ T) \leftarrow\\ &\qquad \textbf{happensAt}(E_{T_1},\ T),\\ &\qquad [\text{conditions}]\\ &\ldots\\ &\textbf{terminatedAt}(CE,\ T) \leftarrow\\ &\qquad \textbf{happensAt}(E_{T_j},\ T),\\ &\qquad [\text{conditions}]\end{aligned}$$

CE recognition

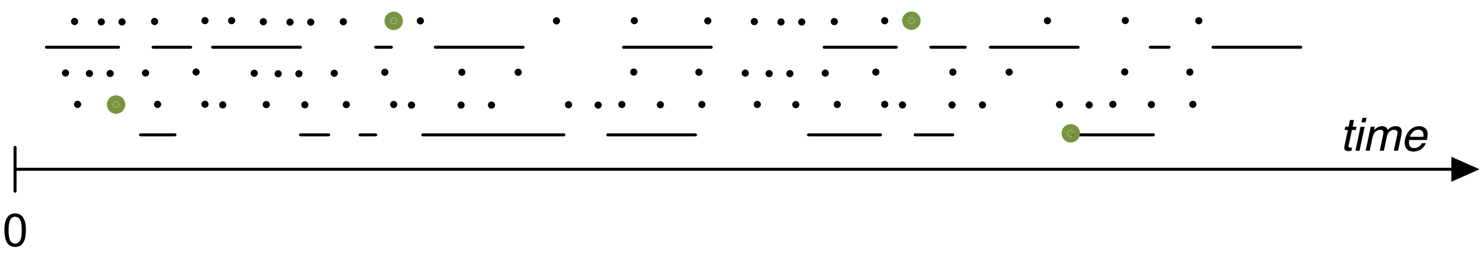


CE Definitions in RTEC: Simple Fluents

CE definition:

$$\begin{array}{ll}
\textbf{initiatedAt}(CE,\ T) \leftarrow & \textbf{terminatedAt}(CE,\ T) \leftarrow \\
\quad \textbf{happensAt}(E_{In_1},\ T), & \quad \textbf{happensAt}(E_{T_1},\ T), \\
\quad [\text{conditions}] & \quad [\text{conditions}] \\
\ldots & \ldots \\
\textbf{initiatedAt}(CE,\ T) \leftarrow & \textbf{terminatedAt}(CE,\ T) \leftarrow \\
\quad \textbf{happensAt}(E_{In_i},\ T), & \quad \textbf{happensAt}(E_{T_j},\ T), \\
\quad [\text{conditions}] & \quad [\text{conditions}]
\end{array}$$

CE recognition:



CE Definitions in RTEC: Simple Fluents

CE definition:

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

...

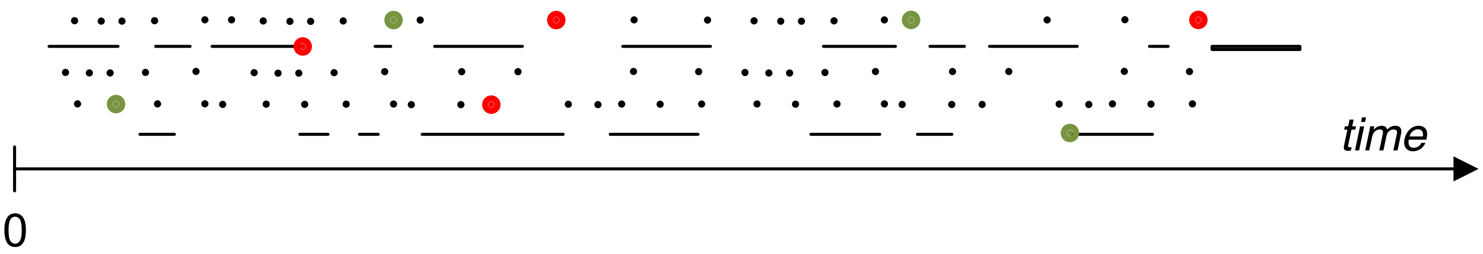
$$\textbf{initiatedAt}(CE,\ T) \leftarrow \\ \quad \textbf{happensAt}(E_{In_i},\ T), \\ \quad [\text{conditions}]$$

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

...

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

CE recognition:



CE Definitions in RTEC: Simple Fluents

CE definition:

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

$\ldots$

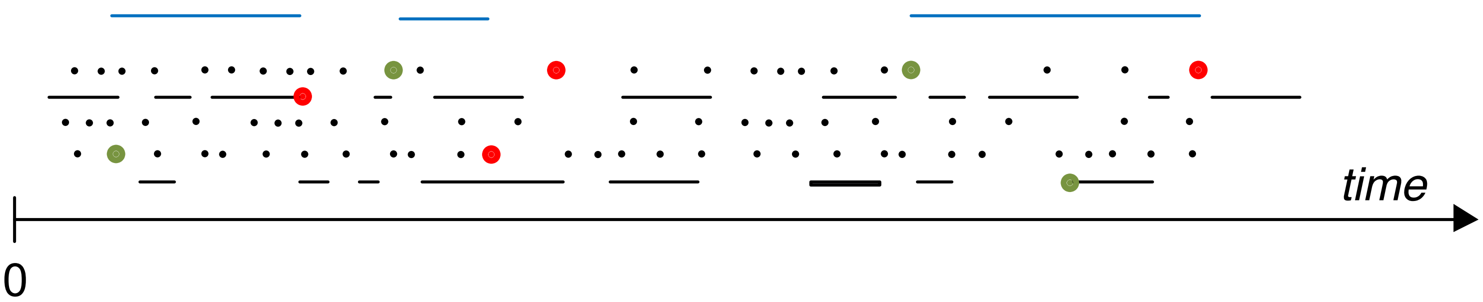
$$\textbf{initiatedAt}(CE,\ T) \leftarrow$$
$$\quad \textbf{happensAt}(E_{In_i},\ T),$$
$$\quad [\text{conditions}]$$

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

$\ldots$

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

CE recognition: **holdsFor**$(CE,\ I)$



CE Definitions in RTEC: Simple Fluents

CE definition:

$$\begin{aligned}
&\textbf{initiatedAt}(\mathit{leaving_object}(P, Obj) = \text{true},\ \ T) \leftarrow \\
&\qquad \textbf{happensAt}(\mathit{appear}(Obj),\ \ T), \\
&\qquad \textbf{holdsAt}(\mathit{inactive}(Obj) = \text{true},\ \ T), \\
&\qquad \textbf{holdsAt}(\mathit{close}(P, Obj) = \text{true},\ \ T), \\
&\qquad \textbf{holdsAt}(\mathit{person}(P) = \text{true},\ \ T)
\end{aligned}$$

$$\begin{aligned}
&\textbf{terminatedAt}(\mathit{leaving_object}(P, Obj) = \text{true},\ \ T) \leftarrow \\
&\qquad \textbf{happensAt}(\mathit{disappear}(Obj),\ \ T)
\end{aligned}$$

CE recognition: $\textbf{holdsFor}(\mathit{leaving\_object}(P, Obj) = \text{true},\ \ I)$

CE Definitions in RTEC: Statically Determined Fluents

$$
\begin{aligned}
&\textbf{holdsFor}(CE,\ I) \leftarrow \\
&\qquad \textbf{holdsFor}(F_1,\ I_{F_1}), \\
&\qquad \ldots, \\
&\qquad \textbf{holdsFor}(F_f,\ I_{F_f}), \\
&\qquad \mathit{interval\_manipulation_1}(I_\alpha, \ldots, I_\omega), \\
&\qquad \ldots, \\
&\qquad \mathit{interval\_manipulation_k}(I_A, \ldots, I_\Omega)
\end{aligned}
$$

where

$$
\begin{aligned}
&\mathit{interval\_manipulation}(I_1, \ldots, I_n,\ I) : \\
&\qquad \mathrm{union}([I_1, \ldots, I_n],\ I) \\
&\qquad \mathrm{intersection}([I_1, \ldots, I_n],\ I) \\
&\qquad \mathrm{relative\_complement}(I_1,\ [I_2, \ldots, I_n],\ I)
\end{aligned}
$$

CE Definitions in RTEC: Statically Determined Fluents

CE definition:

$$
\begin{aligned}
&\mathbf{holdsFor}(abnormal(Vessel) = \mathrm{true},\ I)\ \leftarrow \\
&\qquad \mathbf{holdsFor}(slowMotion(Vessel) = \mathrm{true},\ I_1), \\
&\qquad \mathbf{holdsFor}(gap(Vessel) = \mathrm{true},\ I_2), \\
&\qquad \mathbf{holdsFor}(stop(Vessel) = \mathrm{true},\ I_3), \\
&\qquad \mathrm{union}([I_1, I_2, I_3],\ I)
\end{aligned}
$$

CE Definitions in RTEC: Statically Determined Fluents

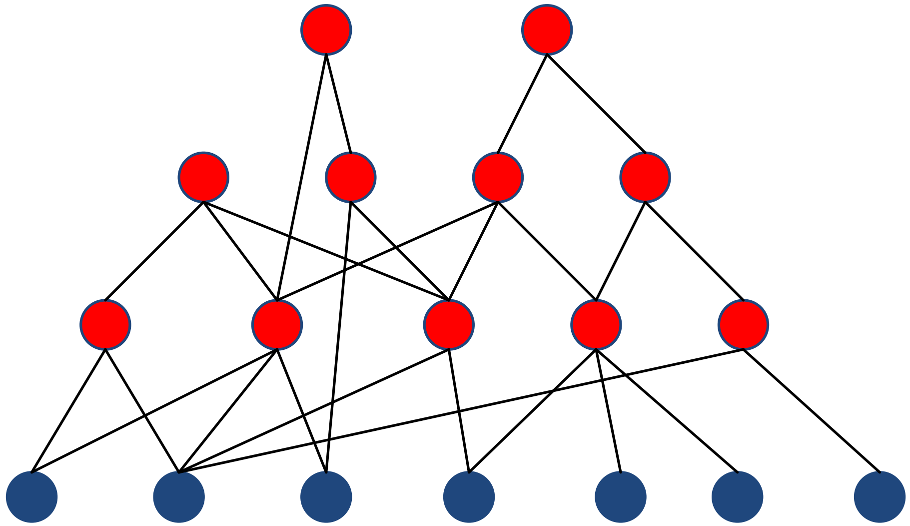
CE definition:

$$
\begin{aligned}
&\textbf{holdsFor}(abnormal(Vessel) = \text{true},\ I) \ \leftarrow \\
&\qquad \textbf{holdsFor}(slowMotion(Vessel) = \text{true},\ I_1), \\
&\qquad \textbf{holdsFor}(gap(Vessel) = \text{true},\ I_2), \\
&\qquad \textbf{holdsFor}(stop(Vessel) = \text{true},\ I_3), \\
&\qquad \text{union}([I_1, I_2, I_3],\ I)
\end{aligned}
$$

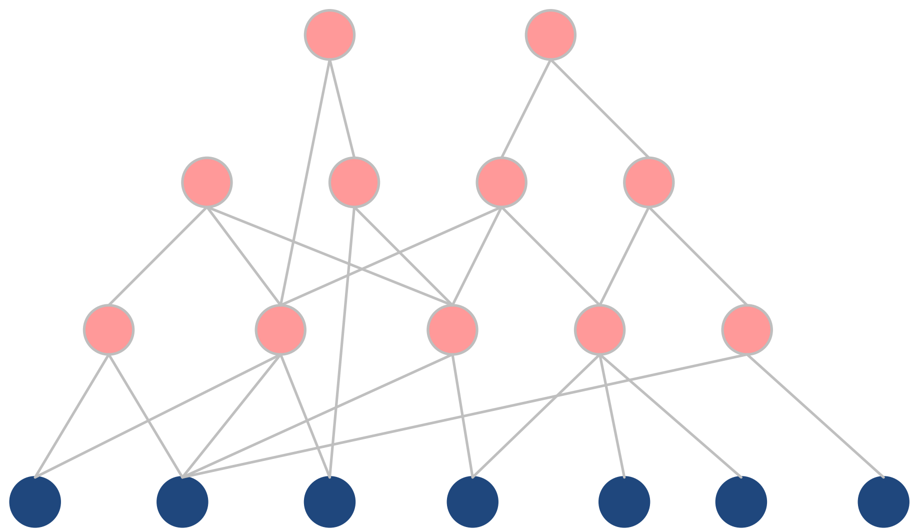
Shorthand:

$$
\begin{aligned}
&abnormal(Vessel) \text{ iff} \\
&\qquad slowMotion(Vessel) \text{ or} \\
&\qquad gap(Vessel) \text{ or} \\
&\qquad idle(Vessel)
\end{aligned}
$$

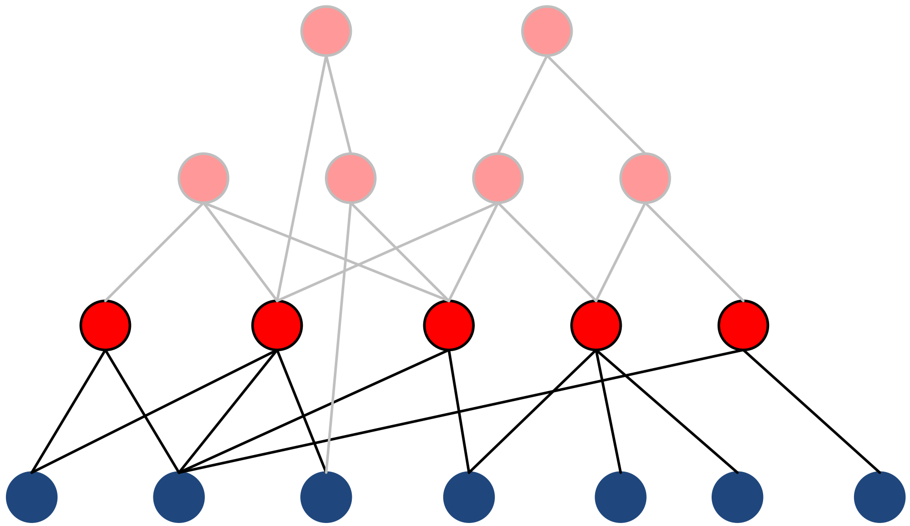
CE Hierarchies



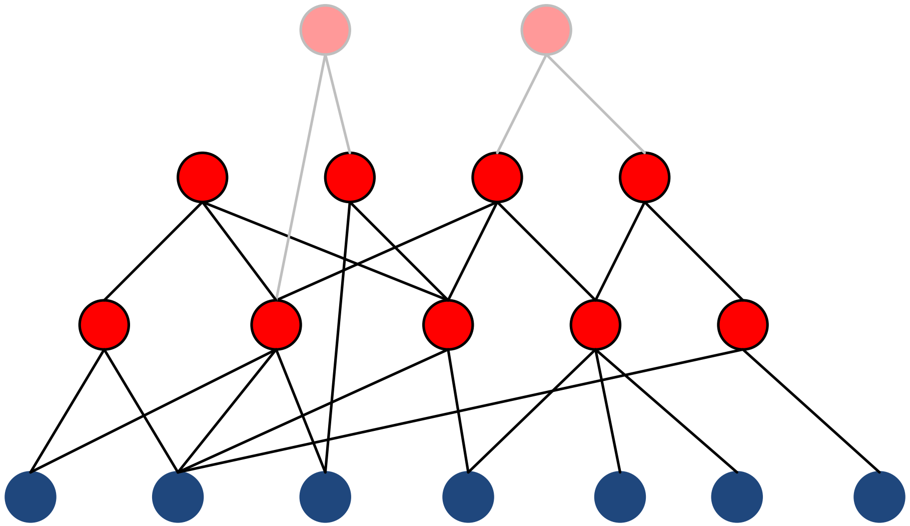
CE Hierarchies: Caching



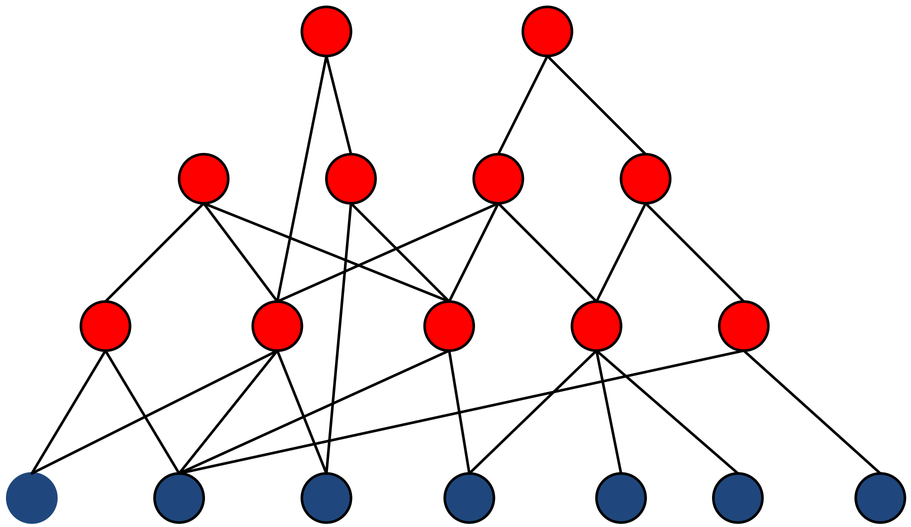
CE Hierarchies: Caching



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

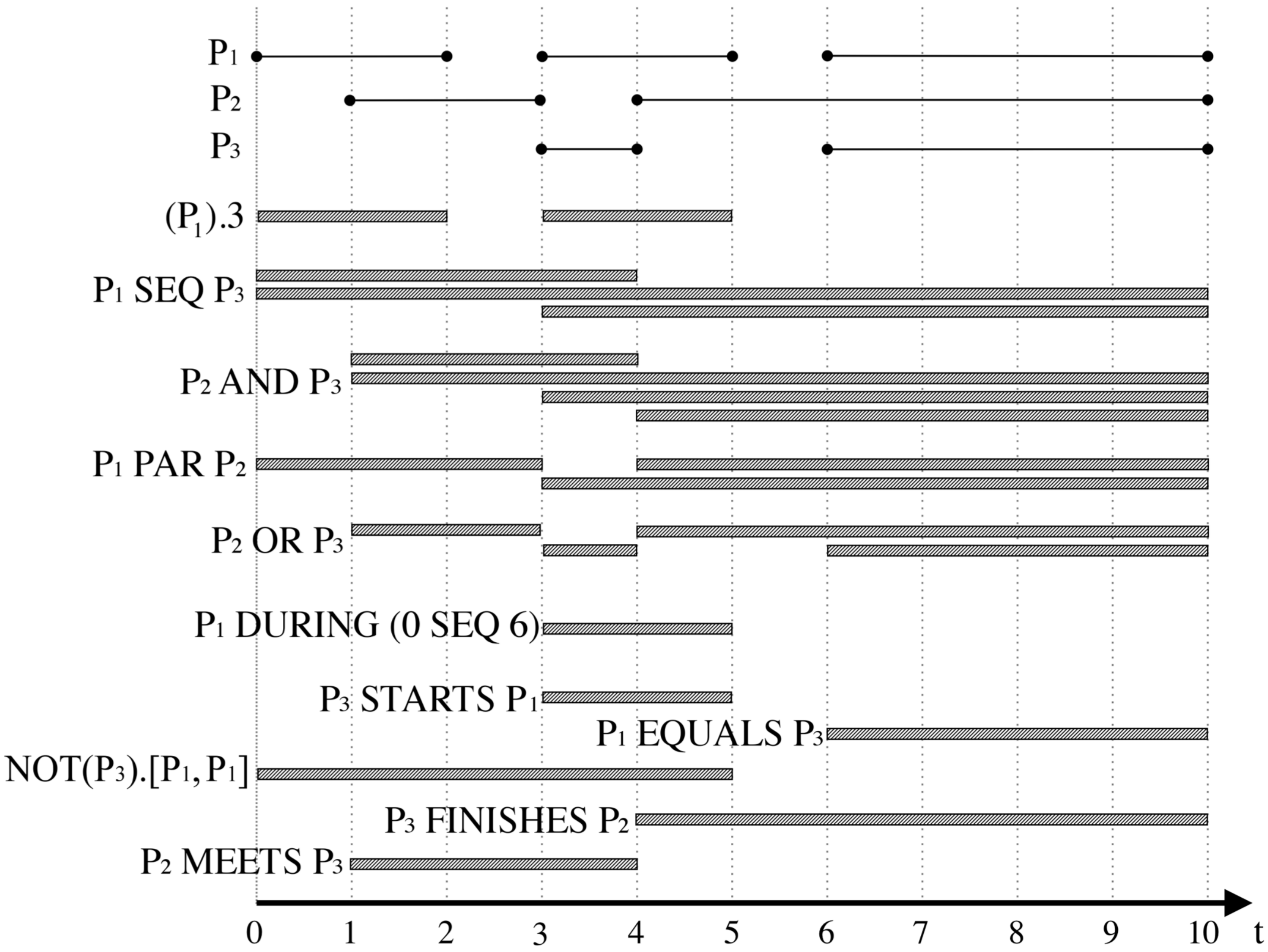
ETALIS: Overview

- 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

$$\text{DangerousAcceleration(X)} \leftarrow$$
$$\qquad \text{Speed(X, Y1) SEQ Speed(X,Y2) WHERE Y2} > \text{Y1} \times 1.5$$

- 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:

$$E \leftarrow E1 \text{ SEQ } C$$

$$E1 \leftarrow A \text{ SEQ } B$$

ETALIS: Recognition

$$E \leftarrow \text{E1 SEQ C}$$

$$E1 \leftarrow \text{A SEQ B}$$

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

ETALIS: Recognition

$$E \leftarrow E1 \text{ SEQ } C$$

$$E1 \leftarrow A \text{ SEQ } B$$

- Transalted 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

$$E \leftarrow E1 \text{ SEQ } C$$

$$E1 \leftarrow A \text{ SEQ } B$$

- Transalted 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?

Outlook for Logic-Based 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?