Garbage Collection for Temporal Stream Algebra (TSA) and Event-Mill

Evgeny Novoseltsev

March 27, 2013

Contents

Introduction

Complex Event Queries

Complex event queries are standing queries that are continosly evaluated against a stream of input events and thereby produce complex output events.

Events

Events = tuples in potentially infinite data streams with finite prefix at each point in time.

Problem

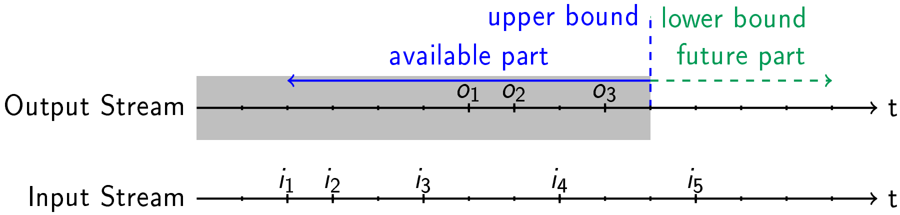
New tuples are continuously buffered at least temporarily. If the buffered tuples are not removed after some time, memory runs full.

The Way to Garbage Collection

We have:

- query with input and output streams.
- progress of the output stream = the upper bound[1]
- available part of the output stream = the part up to the upper bound.

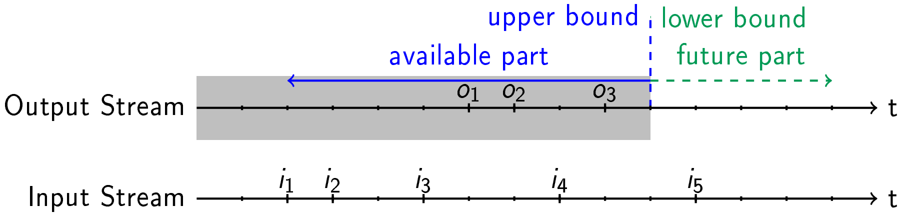
[1] Simon Brodt, Francois Bry. Analysing Temporal Relations.



The Way to Garbage Collection

Goal 1

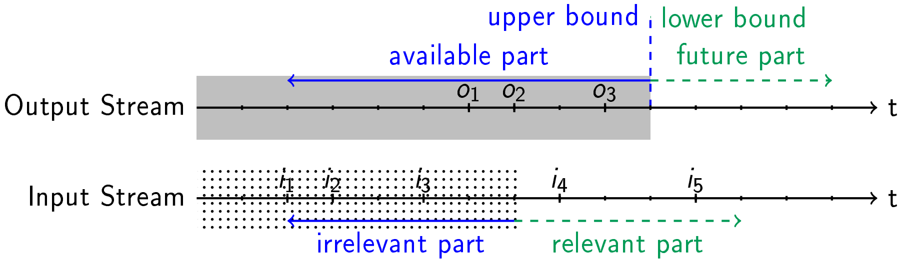
Determing those tuples in the input streams of a query that cannot contribute to further output tuples.



The Way to Garbage Collection

Goal 2

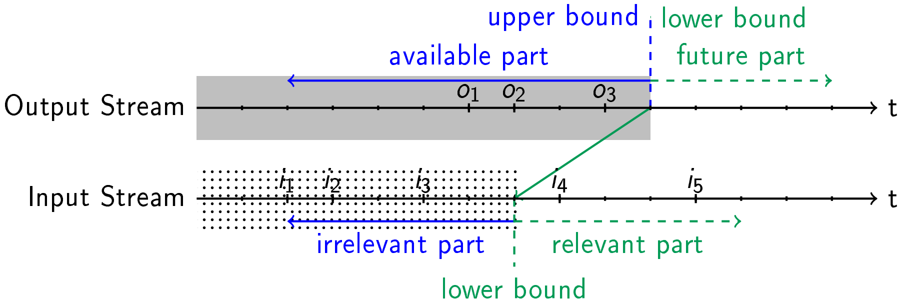
Determining that part of an input stream which contains the irrelevant tuples → selection conditions.



The Way to Garbage Collection

Goal 3

Determining lower bounds of the relevant part of an input stream → values for selection conditions.



Running Example

Running Example - Airport Use-Case

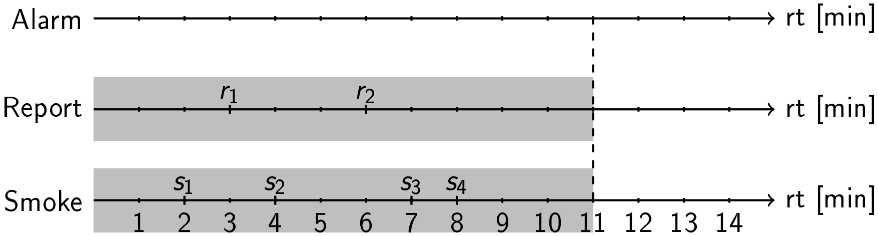
If there is a smoke warning in an area and the responsible warden does not give a report within 5 minutes, then an alarm is raised.

```
DETECT
    Alarm {area {var X}}
ON
    event s: Smoke {area {var X}}
    not event r: Report {area {var X}}
WHERE
    s.rt.end <= r.rt.begin,
    r.rt.begin <= s.rt.end + 5 min
END
```

rt.begin is the beginning, *rt.end* is the end of the reception time interval (commonly known as "system time") of an event.

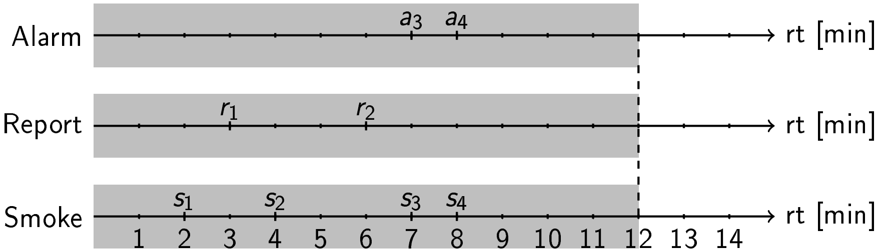
Determining Upper Bounds

Before the 1st step of the naïve evaluation.



Determining Upper Bounds

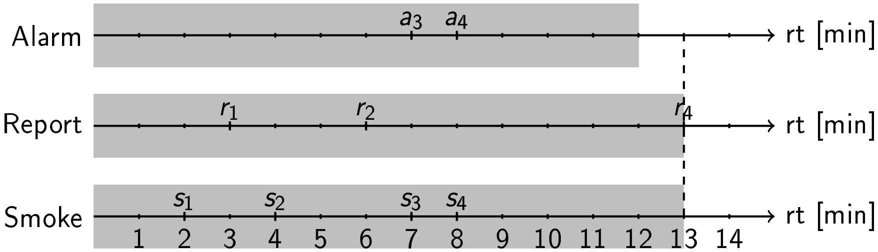
1st step of the naïve evaluation after 12 minutes.



No report following a smoke warning within 5 minutes ⇒ create an alarm.

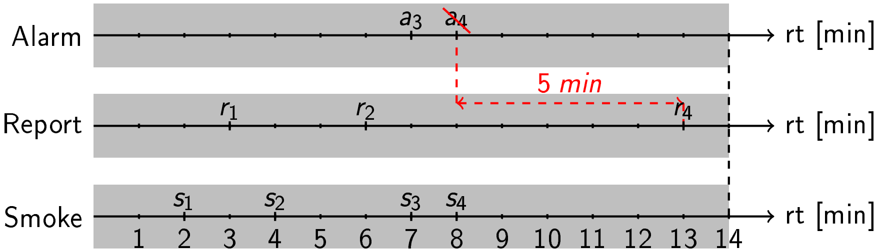
Determining Upper Bounds

let r_4 be a new report after 13 minutes.



Determining Upper Bounds

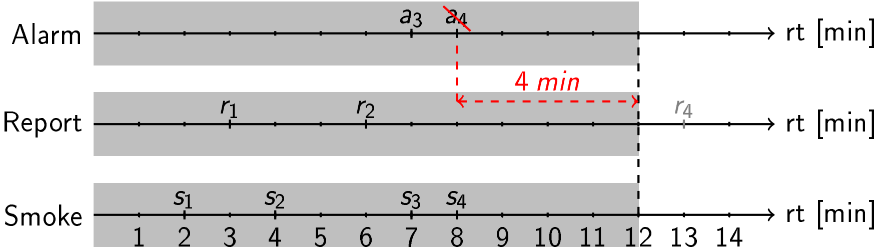
2nd step of the naïve evaluation at $rt = 14$.



$\Rightarrow a_4$ is raised too rashly at time $t = 12\ min$.

Determining Upper Bounds

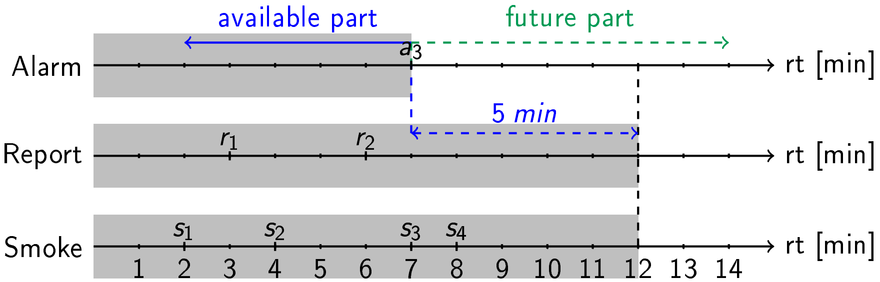
1st step of the naïve evaluation at $rt = 12$.



$\Rightarrow$ Upper bound of *Alarm* = Upper bound of *Report* - 5 Min.

Determining Upper Bounds

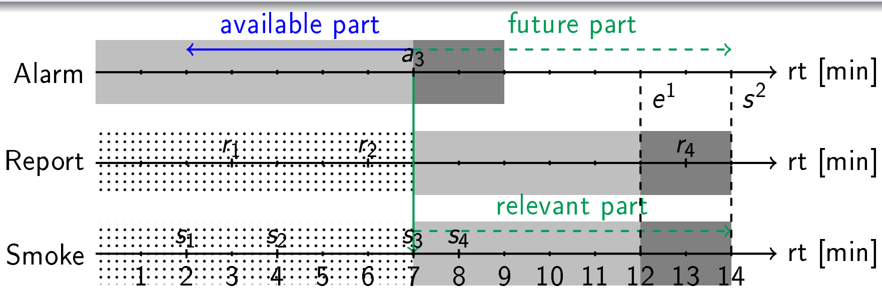
1st step of the incremental evaluation at $rt = 12$ (correct).



Incremental evaluation determines the upper bound of the output stream based on the upper bounds of the input streams.

Determining Lower Bounds

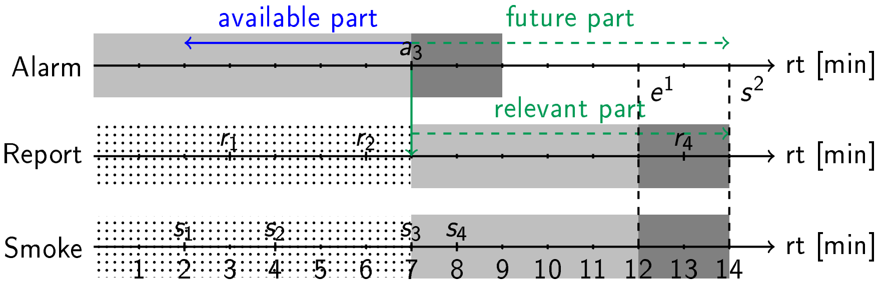
Reception time of the alarm = reception time of the original smoke warning.



Relevance analysis: Lower bounds of the future part of the output stream determine lower bounds to the relevant part of the input streams.

Determining Lower Bounds

A report is only relevant to a smoke warning if it occurs after the warning.



Relevance analysis: Lower bounds of the future part of the output stream determine lower bounds to the relevant part of the input streams.

Relevance Analysis for the Running Example

Algorithm

Goal (at each step of the incremental evaluation)

- Determine the lower bounds for input streams.
- Determine the relevant/irrelevant tuples.

Algorithm

Goal (at each step of the incremental evaluation)

- Determine the lower bounds for input streams.
- Determine the relevant/irrelevant tuples.

Building blocks

1. Keep values with respect to a query/a program (lower bounds).
2. Keep functions (compute keep values for input streams from progress of output stream).
3. Relevance condition (selection condition for relevant tuples based on keep values).
4. Relevance formula (parameterized version of the relevance condition).

Algorithm

Goal (at each step of the incremental evaluation)

- Determine the lower bounds for input streams.
- Determine the relevant/irrelevant tuples.

Building blocks

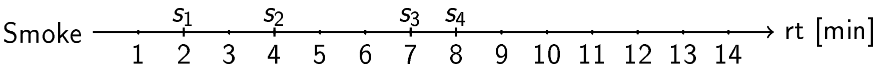
1. Keep values with respect to a query/a program (lower bounds).
2. Keep functions (compute keep values for input streams from progress of output stream).
3. Relevance condition (selection condition for relevant tuples based on keep values).
4. Relevance formula (parameterized version of the relevance condition).

Output: relevance conditions

Only tuples with attribute values greater than the keep values, can contribute to further output tuples.

Relevance Formula

- parametrized version of the relevance condition.
- defined with respect to the timestamp attributes of a temporal stream.



Relevance Formula for *Smoke*

$rt.end > Smoke{:}rt.end$,

where $Smoke{:}rt.end$ is the parameter for a keep value.

Keep Functions

- upper bound of an output $\rightarrow$ lower bound of an input.
- based on the temporal conditions between the attributes

Temporal conditions

$Alarm{:}rt.end < a.rt.end$ - holds for new output tuples

Keep Functions

- upper bound of an output $\rightarrow$ lower bound of an input.
- based on the temporal conditions between the attributes

Temporal conditions

$$Alarm{:}rt.end < a.rt.end$$

$$\left.\begin{array}{l} a.rt.end = s.rt.end \\ s.rt.end \leq r.rt.begin \end{array}\right\} \text{ - query specification}$$

Keep Functions

- upper bound of an output $\rightarrow$ lower bound of an input.
- based on the temporal conditions between the attributes

Temporal conditions

$Alarm{:}rt.end < a.rt.end$
$a.rt.end = s.rt.end$
$s.rt.end \leq r.rt.begin$
$r.rt.begin \leq r.rt.end$ - stream specification

Keep Functions

- upper bound of an output $\rightarrow$ lower bound of an input.
- based on the temporal conditions between the attributes

Temporal conditions

$$\begin{array}{l} \mathit{Alarm}{:}rt.end < a.rt.end \\ \left.\begin{array}{l} a.rt.end = s.rt.end \\ s.rt.end \leq r.rt.begin \end{array}\right\} \Rightarrow a.rt.end \leq r.rt.begin \\ r.rt.begin \leq r.rt.end \end{array}$$

Keep Functions

- upper bound of an output $\rightarrow$ lower bound of an input.
- based on the temporal conditions between the attributes

Temporal conditions

$$\begin{array}{l} Alarm{:}rt.end < a.rt.end \\ \left.\begin{array}{l} \left.\begin{array}{l} a.rt.end = s.rt.end \\ s.rt.end \leq r.rt.begin \end{array}\right\} \Rightarrow a.rt.end \leq r.rt.begin \\ r.rt.begin \leq r.rt.end \end{array}\right\} \Rightarrow a.rt.end \leq r.rt.end \end{array}$$

Keep Functions

- upper bound of an output $\rightarrow$ lower bound of an input.
- based on the temporal conditions between the attributes

Temporal conditions

$$
\left.\begin{array}{l}
\mathit{Alarm}{:}rt.end < a.rt.end \\
\left.\begin{array}{l} a.rt.end = s.rt.end \\ s.rt.end \leq r.rt.begin \end{array}\right\} \Rightarrow a.rt.end \leq r.rt.begin \\
r.rt.begin \leq r.rt.end
\end{array}\right\} \Rightarrow a.rt.end \leq r.rt.end
$$

$$
\Rightarrow \begin{cases} \mathit{Alarm}{:}rt.end < s.rt.end \\ \mathit{Alarm}{:}rt.end < r.rt.end \end{cases}
$$

Keep Functions

- upper bound of an output $\rightarrow$ lower bound of an input.
- based on the temporal conditions between the attributes

Temporal conditions

$$Alarm{:}rt.end < a.rt.end$$

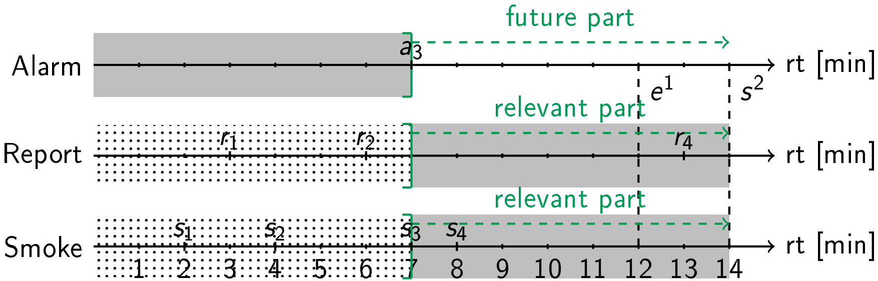
$$\left.\begin{aligned}\left.\begin{aligned}&a.rt.end = s.rt.end\\&s.rt.end \le r.rt.begin\end{aligned}\right\} \Rightarrow a.rt.end \le r.rt.begin\\ r.rt.begin \le r.rt.end\qquad\qquad\end{aligned}\right\} \Rightarrow a.rt.end \le r.rt.end$$

$$\Rightarrow \begin{cases} Alarm{:}rt.end < s.rt.end\\ Alarm{:}rt.end < r.rt.end\end{cases}$$

$$f_{Smoke:rt.end} = f_{Report:rt.end} = Alarm{:}rt.end$$

Keep Values. Beginning of the 2nd Incremental Evaluation

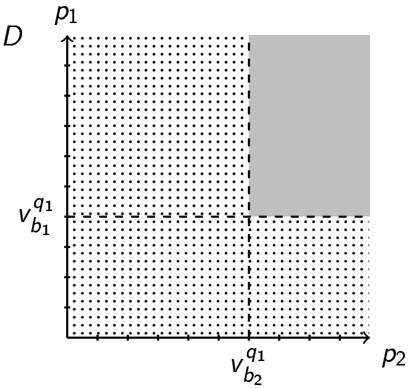
- $f_{Smoke:rt.end} = f_{Report:rt.end} = Alarm:rt.end$
- $Alarm:rt.end = 7\ min$



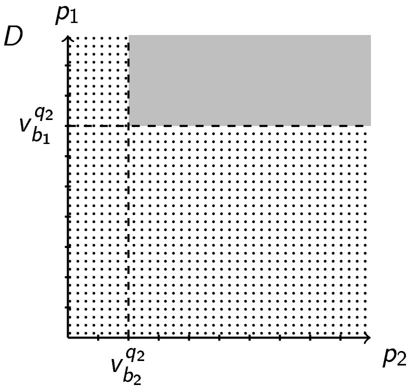
Relevance with Respect to a Program

Relevant tuples for a program

Tuples that are relevant for at least one query in the program.



$$p_1 > v_{b_1}^{q_1} \wedge p_2 > v_{b_2}^{q_1}$$



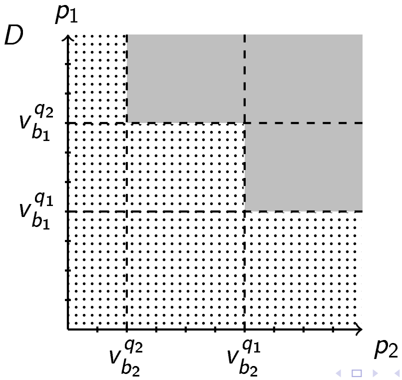
$$p_1 > v_{b_1}^{q_2} \wedge p_2 > v_{b_2}^{q_2}$$

Relevance with Respect to a Program

First approach: Disjunction of the relevance conditions

$$(p_1 > v_{b_1}^{q_1} \ \wedge \ p_2 > v_{b_2}^{q_1}) \vee (p_1 > v_{b_1}^{q_2} \ \wedge \ p_2 > v_{b_2}^{q_2})$$

precisely, but time-consuming.

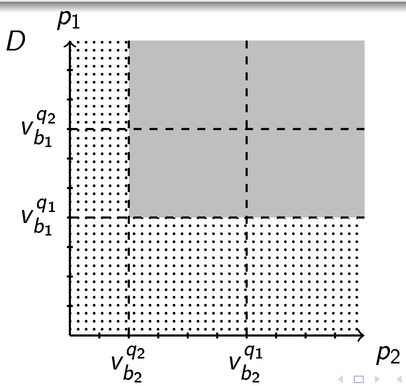


Relevance with Respect to a Program

Second approach: Keep values with respect to a program

$$(p_1 > v_{b_1}^{q_1} \wedge p_2 > v_{b_2}^{q_1}) \vee (p_1 > v_{b_1}^{q_2} \wedge p_2 > v_{b_2}^{q_2}) \Rightarrow$$
$$p_1 > \min\{v_{b_1}^{q_1}, v_{b_1}^{q_2}\} \wedge p_2 > \min\{v_{b_2}^{q_1}, v_{b_2}^{q_2}\}$$

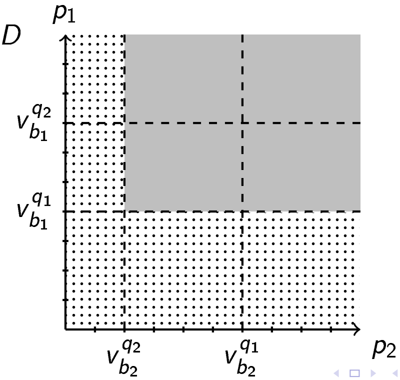
with irrelevant tuples, but non time-consuming (our recommendation)



Garbage Collection

GC has to delete the tuples that are irrelevant for the program, i.e. for all queries ⇒ negation of the relevance conditions with respect to the program:

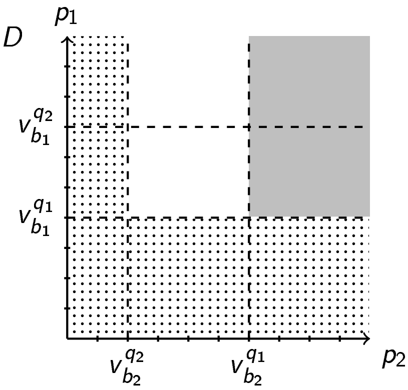
$$p_1 \leq \min\{v_{b_1}^{q_1}, v_{b_1}^{q_2}\} \ \vee \ p_2 \leq \min\{v_{b_2}^{q_1}, v_{b_2}^{q_2}\}$$



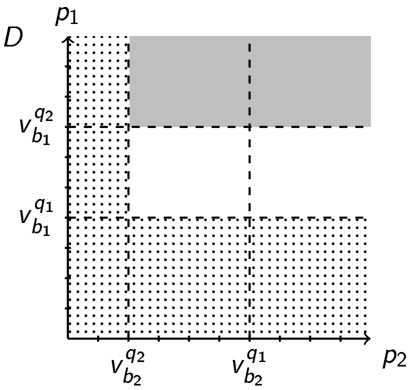
Query Optimization

After execution of garbage collection

- gray areas - relevant tuples for the queries
- white areas - relevant tuples for the program, but not for the queries



$p_1 > v_{b_1}^{q_1} \wedge p_2 > v_{b_2}^{q_1}$



$p_1 > v_{b_1}^{q_2} \wedge p_2 > v_{b_2}^{q_2}$

Definitions

Temporal Stream Algebra (TSA), Informal

TSA Operators

- Standard operators of Relational Algebra: Cross product $\times$, Selection σ, Set Difference $\setminus$ etc.
- Feature: propagation of temporal relations in the stream schema.

Temporal Stream Algebra (TSA), Informal

TSA Operators

- Standard operators of Relational Algebra: Cross product $\times$, Selection σ, Set Difference $\setminus$ etc.
- Feature: propagation of temporal relations in the stream schema.

Temporal Relation Formulas - TRFs

- TRFs describe temporal relations between schema attributes.
- $\perp, \top, t_1 \; op \; t_2, G_1 \wedge G_2, G_1 \vee G_2, \neg G$ are TRFs.
- $t, t + c, min\{t_1, \ldots, t_n\}, max\{t_1, \ldots, t_n\}$ are temporal terms.

Temporal Stream Algebra (TSA), Informal

TSA Operators

- Standard operators of Relational Algebra: Cross product $\times$, Selection σ, Set Difference $\setminus$ etc.
- Feature: propagation of temporal relations in the stream schema.

Temporal Relation Formulas - TRFs

- TRFs describe temporal relations between schema attributes.
- $\perp, \top, t_1 \; op \; t_2, G_1 \wedge G_2, G_1 \vee G_2, \neg G$ are TRFs.
- $t, t + c, min\{t_1, \ldots, t_n\}, max\{t_1, \ldots, t_n\}$ are temporal terms.

Stream bound formulas - SBFs

- SBFs contain the information about attributes, which can be used to obtain a finite prefix of the temporal stream.
- $\top$, $bounded(v, b)$, $H_1 \wedge H_2$, $H_1 \vee H_2$ are SBFs, where $v \in ATTR \cup VAR$, $b \in BOUND$ is a stream bound identifier.

Relevance Formula

Example

Let $H = bounded(p_1, b_1) \vee \ldots \vee bounded(p_n, b_n)$. The relevance formula for H is $\not\triangle H = p_1 > b_1 \wedge \ldots \wedge p_n > b_n$.

Definition

Let H be a SBF. The *relevance formula* $\not\triangle H \triangleq \neg \triangle H$ to H is:

$$
\begin{aligned}
\not\triangle \top &= \bot \\
\not\triangle\, bounded(p, b) &= p > b \\
\not\triangle(H_1 \wedge \ldots \wedge H_k) &= \not\triangle H_1 \vee \ldots \vee \not\triangle H_k \\
\not\triangle(H_1 \vee \ldots \vee H_k) &= \not\triangle H_1 \wedge \ldots \wedge \not\triangle H_k
\end{aligned}
$$

where $p \in ATTR \cup VAR$ and $b \in BOUND$.

Keep Functions

The *keep function* for the stream bound identifier b_E and the input $D_{q,j}$ on position j of the TSA query $q = (D, E)$ is:

$$
\begin{aligned}
f_{b_E,q,j} &= f_{\not\exists H_D, G_E, v} \\
f_{\not\exists H_D, G_E, v} &= \min_{C_E \in dnf(\overline{G_E})} \{ f_{\not\exists H_D, C_E, v} \} \\
f_{(\not\exists H_{D_1} \wedge \ldots \wedge \not\exists H_{D_k}), C_E, v} &= \min(f_{\not\exists H_{D_1}, C_E, v}, \ldots, f_{\not\exists H_{D_k}, C_E, v}) \\
f_{(p > b_D), C_E, v} &= \begin{cases} +\infty & \text{if } \ dist_{C_E}(v, p) = -\infty \\ -\infty & \text{if } \ dist_{C_E}(v, p) = +\infty \\ b_D - dist_{C_E}(v, p) & \text{else} \end{cases}
\end{aligned}
$$

where $bounded(v, b_E)$ occurs in H_E.

Conclusion

Relevance analysis for complex event queries specified in TSA

- Based on in-depth analysis of temporal relations at compile-time (in contrast to user-defined time-windows or consumption policies)
- Keep functions and relevance formulas determined at compile-time
- Keep values and relevance conditions efficiently determined at runtime

Improvements

- Garbage collection preventing memory leaks
- Optimization of incremental evaluation by minimizing the number of irrelevant tuples taking part in the evaluation

Future work

- Performance evaluation of the approach based on a benchmark
- Relevance and garbage collection for TSA queries beyond temporal relations