LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK
LEHR- UND FORSCHUNGSEINHEIT FÜR
PROGRAMMIER- UND MODELLIERUNGSSPRACHEN

Institut für Informatik

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

Evgeny Novoseltsev

Bachelorarbeit

Beginn der Arbeit:	03. Dezember 2012
Abgabe der Arbeit:	28. März 2013
Betreuer:	Prof. Dr. François Bry
	Simon Brodt

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

München, den 28. März 2013 .....

Evgeny Novoseltsev

Zusammenfassung

Die inkrementelle Auswertung von Temporal Stream Algebra (TSA) muss zumindest einen Teil der Eingabetupel einer Anfrage oder eines Programs zeitweise speichern. Diese Bachelorarbeit beschreibt eine Analyse, die es möglich macht zu unterscheiden welche Eingabetupel noch zu weiteren Ausgabetupeln beitragen können, also relevant sind, und welche Eingabetupel nicht zu weiteren Ausgabetupeln beitragen können, also irrelevant sind. Diese Relevanzanalyse basiert auf der Analyse der Zeitbeziehungen, wie sie als Mittel zur Inkrementellen Auswertung in früheren Arbeiten zu TSA beschrieben wurde. Die Arbeit zeigt wie die Relevanzanalyse für die Garbage-Collection benutzt werden kann und so verhindert, dass einem TSA Programm der Speicher ausgeht. Außerdem liefert die Relevanzanalyse sogenannte Relevanz Filter, die benutzt werden können um die inkrementelle Auswertung von TSA Anfragen zu optimieren, indem die Zahl der an der Auswertung beteiligten irrelevanten Tupel minimiert wird.

Abstract

The incremental evaluation of Temporal Stream Algebra (TSA) needs to buffer at least parts of the input tuples of a query or program for some time. This bachelor thesis introduces an analysis which allows to distinguish input tuples that can contribute to further output tuples, i.e. are *relevant*, from those tuples that cannot contribute to any further output tuple, i.e. are *irrelevant*. This *relevance analysis* bases on the analysis of temporal relations which is described in prior work on TSA as a mean for enabling the incremantal evaluation of TSA. The thesis shows how the relevance analysis can be used for garbage collection, preventing a TSA program from running out of memory. Furthermore the relevance analysis yields so-called *relevance filters* which can be used to optimze the incremental evaluation of TSA queries by minimizing the number of irrelevant input tuiples taking part in the incremental evaluation.

Contents

1 Introduction

Emergency management in large infrastructures requires a timely and efficient detection of emergencies and critical situations based on data from sensors. The precise detection of a critical situation requires a combination of data from multiple sensors, e.g. a fire alarm in an area should be raised if a temperature sensor signals a high temperature and a smoke detector signals a significant smoke concentration within one minute. Complex Event Processing (CEP) is a method to detect such patterns. CEP allows to formulate rules that derive complex events from specific combinations or patterns of simple events.

The "Declarative uniform reactive event processing language" (Dura, [SH11]) is an event query languauge that has been designed for meeting the requirements of emergency management. Event-Mill is the evaluation engine for Dura. Event-Mill bases on Temporal Stream Algebra (TSA, [SB13]) that generalizes the data model and the standard operators of relational algebra towards data streams.

The incremental evaluation of TSA queries needs to buffer at least parts of the input events/-tuples from the input streams for some time. If the stored tuples are not removed after some time, memory obviously runs full. The question is which tuples must be kept or deleted. Tuples that cannot contribute to further output tuples in any TSA query of a TSA program are irrelevant and can be deleted (e.g. by garbage collection). Relevant tuples can contribute to further output tuples in at least one TSA query and must be kept. Tuples that are relevant for one TSA query can be irrelevant for other TSA queries, but at the first place all available tuples from the input streams take part in the incremental evaluation of a TSA query and may significantly but unnecessarily increase intermediate results. Thus filters at the inputs of a TSA query that minimize the number of irrelevant tuples taking part in the incremental evaluation of that query are very important for the efficiency of the incremental evaluation. This bachelor thesis describes an algorithm for determining irrelevant input tuples for single TSA queries as well as for full TSA programs (relevance analysis).

The remaining part of this bachelor thesis is structured as follows. Section 2 introduces the running example of this bachelor thesis and motivates the presented approach towards relevance analysis and garbage collection based on this example. Section 3 gives a formal description of relevance analysis, query optimization, and garbage collection. Section 4 applies the relevance analysis to the running example determining the relevance conditions for optimization (relevance filters) and garbage collection. Section 5 concludes this bachelor thesis.

2 Motivation

2.1 Running Example

The Running Example comes from the airport use-case of the EMILI-project[1] (Emergency Management In Large Infrastructures). The goal of the Running Example is the precise

[1] http://www.emili-project.eu

detection of smoke. Smoke detectors often raise false pre-alarms. For this reason, there is a responsible warden that checks the area of the smoke pre-alarm. If there is no smoke in the area, it sends a report to invalidate the previously smoke pre-alarm. Otherwise, a precautionary alarm is raised. The formal description of the Running Example is given below:

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

Figure 1 shows the Dura query of the example. There are two types of incoming events, which are stored as tuples in the following input streams: *PreAlarm* and *Report*. Output events are inserted into *PcAlarm*. Each tuple has three attributes: *area*, *rt.begin* and *rt.end*. The attribute *area* specifies the place of the event, the time attributes *rt.begin* and *rt.end* specify the beginning and the end of the reception time (system time) of the event.

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

Figure 1: Dura query for the example 2.1

2.2 Upper bounds - Incremental Evaluation

In the following, the example from Section 2.1 illustrates the detection of precautionary alarms based on existing data in the data streams *PreAlarm* and *Report*, which are shown in Figure 2. As to keep the example simple, the value of *rt.begin* is the same as the value of *rt.end* for each tuple, but generally, these values can be different. Figure 3 shows the

PreAlarm	id	area	rt.begin	rt.end
	a_1	1	2	2
	a_2	1	4	4
	a_3	1	7	7
	a_4	1	8	8

Report	id	area	rt.begin	rt.end
	r_1	1	3	3
	r_2	1	6	6

Figure 2: Input streams

data in the output stream *PcAlarm* after the first naive evaluation of the query at time $t = 12\,min$. This incorrect result arises from the naive evaluation of the query by taking all available data from the input streams. Figure 4 shows the incorrect result of the first naive evaluation of the query with the available part of each data stream. The available parts are shown as a gray area, each time unit on the *rt*-axis represents one minute.

PcAlarm	id	area	rt.begin	rt.end
	p_3	1	7	7
	p_4	1	8	8

Figure 3: Output stream after the first naive evaluation

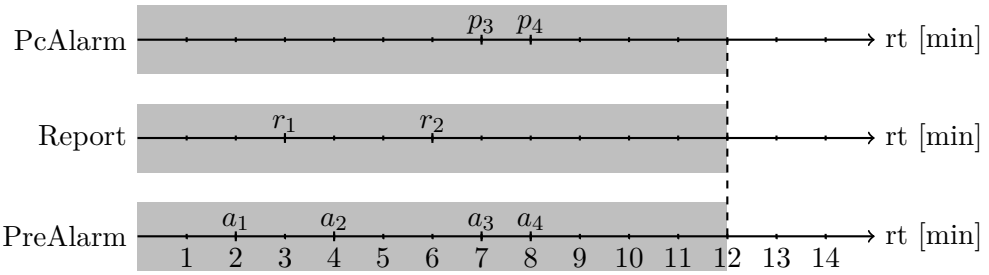

Figure 4: Output and input streams after the first naive evaluation (incorrect)

Let a new report r_4 arrive at time $t = 13$ and let the query be evaluated at time $t = 14\,min$. Figure 5 shows the data in the output stream *PcAlarm*. Figure 6 shows the available parts of all streams after the second naive evaluation.

PcAlarm	id	area	rt.begin	rt.end
	p_3	1	7	7

Figure 5: Output stream after the second evaluation

There is only one precautionary alarm p_3 in the output stream after the second naive evaluation, although there was two precautionary alarms p_3 and p_4 in the output stream after the first naive evaluation (Figure 3 and 4). This means the precautionary alarm p_4 is raised too rashly at time $t = 12\,min$, because the report r_4 still had one minute to fulfill the inequality $r.rt.end <= a.rt.end + 5min$ and thereby to invalidate the pre-alarm a_4. Therefore, a part of the output stream with the precautionary alarm p_4 must not occur in the available part of the output stream after the first evaluation of the query. The problem consists in the determination of the part of the output stream that has to be omitted by an evaluation of the query. The following analysis of temporal relations between attributes in output and input streams shows how to omit the part of the output stream, which contains only the events that are rased too rashly (e.g. the event p_4).

In our example, the output stream *PcAlarm* contains precautionary alarms for the pre-alarms from the input stream *PreAlarm* that have not been invalidated by the following reports from the input stream *Report*. From this, following important conclusions can be drawn.

Firstly, a precautionary alarm semantically occurs at the same time as the original pre-alarm, e.g. $p_3.rt.begin = a_3.rt.begin$ and $p_3.rt.end = a_3.rt.end$. Secondly, it has to be guaranteed that a precautionary alarm can not be raised within 5 minutes prior to a report. This implies that the upper bound of the available part of the output stream *PcAlarm* has

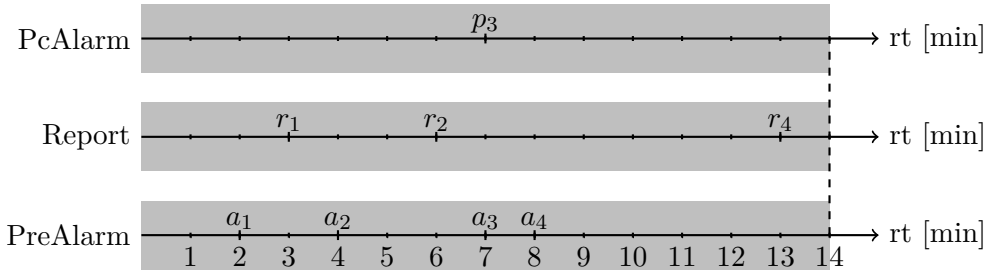

Figure 6: Output and input streams after the second naive evaluation

to be limited to a 5 minute delay compared to the upper bound of the available part of the input stream *Report*.

Figure 7 shows the correct available parts of all data streams after the first incremental evaluation at time $t = 12\,min$.

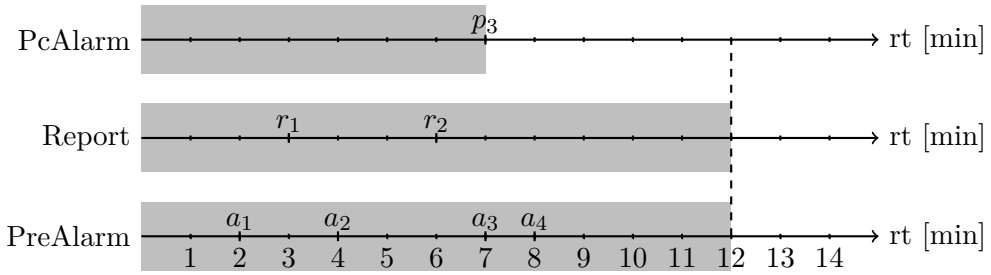

Figure 7: Output and input streams after the first incremental evaluation

The formal algorithm for computation of upper bounds[2] for an incremental evaluation is described in [SB13], Section "Incremental Evaluation".

2.3 Lower Bounds - Garbage Collection

New tuples continuously arrive on data streams and need to be stored at least temporarily. If the stored tuples are not removed after some time, memory obviously run full. At each point in time some of the stored tuples are still relevant to some queries, but most of the stored tuples can not contribute to further output tuples in any query and can be deleted.

The incremental evaluation allows to compute output tuples continuously as input tuples arrive on the input streams. At each step of the incremental evaluation of a query the progress of an output stream from the last incremental evaluation is known. This current progress of an output stream is basically the maximum point in time for which all tuples of the output stream up to that point in time have already been derived. All new output tuples in the future refer to a later point in time. Knowing the progress of the output stream, it is possible to determine input tuples which can certainly not contribute to any further output tuples (i.e. irrelevant input tuples). This process is called relevance analysis and bases on

[2]Progress values, [SB13], Definition 26

the analysis of the temporal relations between input and output streams specified in the query. A formal description based on Temporal Stream Algebra (TSA[3]) is given in Section 3.

Coming back to the Running Example, the progress of the input streams *PreAlarm* and *Report* at the first incremental evaluation at time $t = 12\,min$ is $rt = 12\,min$. For each tuple a_i from the input stream *PreAlarm*, such that $a_i.rt.end \leq 12 - 5 = 7$, which have not been invalidated by a following report r_i from *Report*, such that $r_i.rt.begin \leq 12$, a precautionary alarm p_i is created with $p_i.rt.begin = a_i.rt.begin$ and $p_i.rt.end = a_i.rt.end$. The progress of *PcAlarm* after the first incremental evaluation is $rt = 7\,min$. All further output tuples hold $rt > 7$. Therefore, only input tuples a_i from *PreAlarm* such that $a_i.rt.end > 7$ can contribute to further output tuples. Simultaneously, only input tuples r_i from *Report*, such that $r_i.rt.begin > 7$, can invalidate tuples from *PreAlarm* with $a_i.rt.end > 7$.

Figure 8 shows all input tuples for the query at time $t = 14\,min$ of the second incremental evaluation, where the irrelevant tuples are located in dotted, and the relevant are located in gray areas. The two dashed lines show the time of both incremental evaluations e_1 and e_2 of the query.

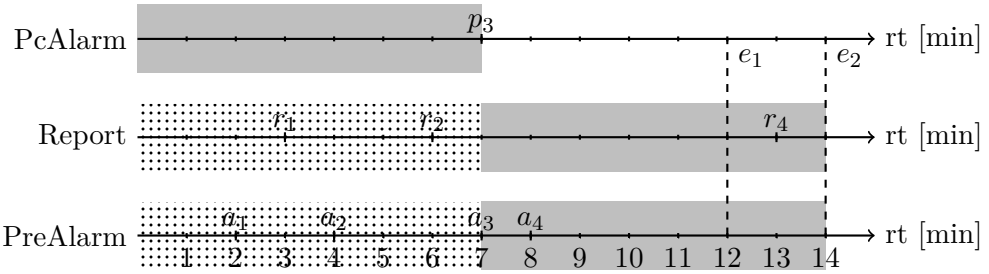

Figure 8: Relevant and irrelevant tuples at the second incremental evaluation

3 Relevance Analysis

Relevance analysis consists in the analysis of temporal relations between input and output streams. The goal of the relevance analysis is to determine the relevant input tuples from each input stream in the TSA program.

3.1 TSA Program

Temporal Stream Algebra (TSA) provides the common concept for events, states and actions defined in a high level programing language like Dura. The concept bases on potentially infinite temporal streams. TSA generalizes the operators of the Relational Algebra for querying streams and the database relations[4]. TSA makes the TSA analysis of temporal relations defined in Dura rules possible. This allows incremental evaluation of the TSA

[3] [SB13]
[4] For details see [SB13]

queries. A TSA program is created from a Dura program with the initialization of the system.

Formally, a TSA program is a pair $P = (T, Q)$, where $T = \{D_1, \ldots, D_n\}$ is the set of temporal stream definitions and $Q = \{q_1, \ldots, q_m\}$ is the set of TSA queries that are equivalent to Dura queries.

In the following, Definitions 1 - 8 from [SB13], which are most relevant for this bachelor thesis, are introduced.

Definition 1 (*Temporal Stream*). [5]

1. A *data stream* R with attribute schema $\mathscr{A}(R) \subseteq ATTR$ ($ATTR$ is a set of attribute names) is a (possibly infinite) relation with finite prefix R^t at each point in time $t \in \mathbb{Q}$, i.e.

   $$R = \bigcup_{t \in \mathbb{Q}} R^t \text{ with } t_1 \leq t_2 \Rightarrow R^{t_1} \subseteq R^{t_2} \text{ and } |R^t| < +\infty \text{ for all } p, p_1, p_2 \in \mathbb{Q}$$

2. R is *static* iff $R^t = R$ for all $t \in \mathbb{Q}$
3. $\emptyset$ is a *progressing set of attributes for* R iff R is static
4. $\{p_1, \ldots, p_k\} \neq \emptyset$ is a *progressing set of attributes for* R iff $\{p_1, \ldots, p_k\} \subseteq \mathscr{A}(R)$ and R eventually exceeds any upper bound $s_1, \ldots, s_k \in \mathbb{Q}$ for $p_1, \ldots, p_k$, i.e.

   $$\exists t \in \mathbb{Q} : \{r \in R \mid r(p_1) \leq s_1 \wedge \cdots \wedge r(p_k) \leq s_k\} \subseteq R^t$$

5. A data stream R is a *temporal stream* iff R has a progressing set of attributes

Definition 2 (*Temporal Terms*). [6] The set of *temporal terms* is defined inductively:

1. $v \in ATTR \cup VAR$ is an atomic temporal term
2. $t + c$ is a temporal term
   iff t is a temporal term and $t \in \mathbb{Q}$
3. $min\{t_1, \ldots, t_k\}$, $max\{t_1, \ldots, t_k\}$ are temporal terms
   iff $t_1, \ldots, t_k$ are temporal terms

where VAR is a set of variables, $VAR \cap ATTR = \emptyset$.

Definition 3 (*Temporal relation formulas*). [7] *Temporal relation formulas* (TRFs) describe temporal relations between temporal terms. The set of temporal relation formulas is defined inductively:

1. $\top$ and $\bot$ are atomic TRFs
2. $t_1 \; op \; t_2$ is an atomic TRF for $op \in \{<, \leq, =, \geq, >, \neq\}$
   iff t_1 and t_2 are temporal terms
3. $G \wedge G'$, $G \vee G'$ and $\neg G$ are TRFs iff G, G' are TRFs

Definition 4 (*Stream Bound Formulas*). [8] *Stream Bound Formulas* (SBFs) contain the

[5] Copied from Definition 1, [SB13]
[6] Copied from Definition 2, [SB13]
[7] Copied from Definition 3, [SB13]
[8] Copied from Definition 6, [SB13]

information about attributes, which can be used to obtain a finite prefix of the temporal stream. The set of stream bound formulas is defined inductively:

1. $\top$ is an atomic SBF
2. $bounded(v, b)$ is an atomic SBF for $v \in ATTR \cup VAR$ where $b \in BOUND$ is a stream bound identifier
3. $H_1 \wedge H_2$ and $H_1 \vee H_2$ are SBFs iff H_1 and H_2 are SBFs

Definition 5 (*Temporal Stream Schema*)**.** [9]

1. A *temporal stream schema* is a triple $S = (A, G, H)$ where $A \subseteq ATTR$ is an attribute schema, G is a TRF with $attr(G) \subseteq A$ and H is a SBF with $attr(H) \subseteq A$
2. S is *static* iff H is equivalent to $\top$
3. $\{p_1, \ldots, p_k\} \subseteq A$ is a *progressing set of attributes* for $S = (A, G, H)$ iff $\{p_1, \ldots, p_k\}$ is a progressing set of attributes with respect to G and H
4. S is *valid* iff S has a progressing set of attributes
5. A relation R matches a temporal stream schema $S = (\mathscr{A}, G, H)$ if R has attribute schema $\mathscr{A}(R) = \mathscr{A}$ and both G and H hold in R.

Definition 6 (*Temporal Stream Definition*)**.** [10] A *temporal stream definition* is a pair $D = (n, S)$ where $n \in STREAM$ is a name for the temporal stream definition, $S = (\mathscr{A}, G, H)$ is a temporal stream schema, $STREAM$ is a set of names for temporal stream definitions, and the following restrictions hold:

1. G and H do not contain variables
2. H is a disjunction of atomic SBFs, i.e. has the structure

   $bounded(p_1, b_1) \vee \ldots \vee bounded(p_k, b_k)$

   where $p_1, \ldots, p_k \in \mathscr{A}$ and $b_1, \ldots, b_k \in BOUND$
3. The atomic SBFs $bounded(p_i, b_i)$ in H define an injection $b^D : \mathscr{A} \to BOUND$ with inverse p^D

Definition 7 (*TSA Expression*)**.** The set of *TSA expressions* is defined inductively:

1. Each temporal stream definition D is a TSA Expression
2. $op(E_1, \ldots, E_n)$ is a TSA expression
   iff $E_1, \ldots, E_n$ are TSA expressions and op is a TSA operator[11]

The schema of a TSA expression can contain variables, which are created by discarding some attributes.

Definition 8 (*TSA Query*)**.** [12] A TSA query is a pair $q = (D, E)$ such that E is a TSA expression with schema $\mathcal{S}(E) = (\mathscr{A}, G_E, H_E)$ and D is a temporal stream definition for the output stream with schema $\mathcal{S}(D) = (\mathscr{A}, G_D, H_D)$ which both have the same attributes.

[9] Copied from Definition 8, [SB13]
[10] Copied from Definition 11, [SB13]
[11] The TSA operators are defined in [SB13], in Section "Operators Reloaded" and used in this bachelor thesis in Section 4.1
[12] Copied from Definition 19, [SB13]

Each TSA query has a set of input temporal stream definitions. A TSA expression of a query is constructed by applying TSA operators to input temporal stream definitions. A schema of a TSA expression is created through the propagation of TRFs and SBFs from input temporal stream definitions. The creation of a TSA expression with its schema for the example from Section 2.1 is given in Section 4.1.

The following Definitions are done in the bachelor thesis in order to accomplish the relevance analysis.

Definition 9 specifies the arity of a TSA query (the amount of input temporal stream definitions in the query). Definition 10 specifies indexed input, where the index of an input specifies the position of the input within the TSA query.

Definition 9 (*Arity of a TSA Query*)**.** Let $q = (D, E)$ be a TSA query. The *arity of the TSA query* q is defined recursively:

1. $arity(q) = arity(E)$
2. $arity(D') = 1$ for a temporal stream definition D'
3. $arity(op(E_1, \ldots, E_n)) = \sum_{i=1}^{n} arity(E_i)$

Definition 10 (*Indexed input of a TSA query*)**.** Let $q = (D, E)$ be a TSA query. The input $D_{q,i} = input_q(i)$ on position i in the query q is defined recursively:

1. $input_q(i) = input_E(i)$
2. $input_{D'}(1) = D'$ for a temporal stream definition D'
3. $input_{op(E_1,\ldots,E_n)}(i) = input_{E_j}(i - \sum_{k=1}^{j-1} E_k)$, where $\sum_{k=1}^{j-1} E_k < i \leq \sum_{k=1}^{j} E_k$

The following lemma is based on the method of construction of TSA expressions and the operating principle of TSA operators. A temporal stream definition can occur several times within a query on different positions, i.e. the temporal stream definition is taken by several TSA operators occurring in the TSA expression. Each TSA operator, applied to several temporal stream definitions, conjugates the SBFs of the temporal stream definitions, although the attributes occurring in SBFs can be replaced by variables.

Lemma 1. Let $q = (D, E)$ be a TSA query with $arity(q) = n$. The SBF of the TSA expression E is representable as a conjunction:

$$H_E = H_{I_1} \wedge \ldots \wedge H_{I_n}$$

where $H_{I_j} = \xi_j(H_{D_{q,j}})$, $H_{D_{q,j}}$ is the SBF of the input of the query q on position j and $\xi_j : a \mapsto v$ is an injective substitution[13], that replaces attributes $a \in \mathscr{A}(D_{q,j})$ by variables $v \in ATTR \cup VAR$, such that for each $a, a' \in \bigcup_j \mathscr{A}(D_{q,j})$ and $l \neq k$ holds $\xi_l(a) \neq \xi_k(a')$, $j = 1, \ldots, n$, $1 \leq l, k \leq n$.

[13]The description of the substitution is taken from [SB13], Definitions 14, 15, 17, 18

3.2 Algorithm

The algorithm for computing the relevant tuples for a TSA program must yield those tuples, which can contribute to further output tuples in the output temporal stream of a query, from each input temporal stream at run time. This section describes the algorithm and provides the relevant definitions. The algorithm is composed of several steps.

In principle, for each input temporal stream definition corresponding to an input temporal stream, a *relevance condition* is determined, which is applied to the input temporal stream and thereby yields the relevant tuples for the TSA program at run-time. The relevance condition for a temporal stream definition is based on a *relevance formula* that is a parametrized version of the relevance condition. A relevance condition can be obtained from a relevance formula by replacing each parameter $b \in BOUND$ by its *keep value* computed at run-time. The relevace conditions says that only tuples with attribute values greater than the keep values, can contribute to further output tuples.

The keep values are computed by *keep functions* based on the temporal relations between attributes, which are specified in the queries of a TSA query. The keep functions compute the keep values of a temporal stream definition with respect to an indexed input of a query based on the current progress of the query. The current progress of a query is reflected by so-called *progress values*[14].

Step 1. Relevance formula.

The *relevance formula* is defined for each temporal stream definition.

Definition 11 (*Relevance Formula*)**.** Let H be a SBF. The *relevance formula* $\not\triangle H \stackrel{\triangle}{=} \neg \triangle H$ to H is the negation of the *pass formula*[15] and it is defined recursively:

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

where $p \in ATTR \cup VAR$ and $b \in BOUND$. $\not\triangle H$ results from H by negating H, pushing the negation down to the atoms using De Morgan's laws, and replacing each negated atom $\neg bounded(p,b)$ in H by $p > b$.

Each stream bound identifier $b \in BOUND$ in a relevance formula is replaced with its *keep value* at run-time in order to obtain a relevance condition. The keep values are computed by *keep functions* described in the next step.

Step 2. Keep function.

A keep function is based on a *keep function formula*, which specifies temporal conditions between attributes of output and input streams of a TSA query. The keep function formula is defined as follows.

[14] Progress values are defined in [SB13], Definition 26
[15] [SB13], Defiition 24

Let G_E be a TRF, H_I and H_D be SBFs, $b \in BOUND$ the identifier for a stream bound and $p \in ATTR$ be an attribute assigned to b. The keep function formula $F_{\not\triangleleft H_D, G_E, \not\triangleleft H_I}$ for $\not\triangleleft H_D$, G_E and $\not\triangleleft H_I$ is defined as a formula that must fulfill the implication $\neg \not\triangleleft H_I \wedge G_E \Rightarrow \neg \not\triangleleft H_D$:

$$F_{\not\triangleleft H_D, G_E, \not\triangleleft H_I} \Rightarrow (\neg \not\triangleleft H_I \wedge G_E \Rightarrow \neg \not\triangleleft H_D)$$

The meaning of the implication is to provide the condition $F_{\not\triangleleft H_D, G_E, \not\triangleleft H_I}$ between the attributes from H_D and H_I. This condition implies that the input tuples no more relevant for the query due to temporal conditions described in G_E are irrelevant for the output stream.

Thus, for $\not\triangleleft H_D = (p > b_D)$, $\not\triangleleft H_I = (v > b_E)$ and G_E such that $p \leq v + dist_{G_E}(v, p)$, a condition $F_{(p>b_D), G_E, (v>b_E)}$ is needed to fulfill the following statement: if $v \leq b_E$ for an input tuple, then this input tuple can contribute only to output tuples with $p \leq b_D$. Given this condition, the following can be concluded:

$$p \leq v + dist_{G_E}(v, p) \leq b_E + dist_{G_E}(v, p) \overset{!}{\leq} (b_D - dist_{G_E}(v, p)) + dist_{G_E}(v, p) = b_D$$

Therefore, $F_{(p>b_D), G_E, (v>b_E)} = (b_E \leq b_D - dist_{G_E}(v, p))$. If $dist_{G_E}(v, p) = -\infty$, then $p \leq v - \infty = -\infty$ and $-\infty \leq b_D$ is true. If $dist_{G_E}(v, p) = \infty$, then $p \leq v + \infty = \infty$ and $\infty \leq b_D$ is false. This implies:

$$F_{(p>b_D), G_E, (v>b_E)} = \begin{cases} \top & \text{if } dist_{C_E}(v, p) = -\infty \\ \bot & \text{if } dist_{C_E}(v, p) = +\infty \\ b_E \leq b_D - dist_{C_E}(v, p) & \text{else} \end{cases}$$

From Definition 12.4 and 12.3 the keep function for the query q and b_E is:

$$f_{b_E, q} = f_{\not\triangleleft H_D, G_E, \not\triangleleft H_I} = f_{F_{\not\triangleleft H_D, G_E, \not\triangleleft H_I}} = f_{b_E \leq b_D - dist_{G_E}(v,p)} = b_D - dist_{G_E}(v, p)$$

The construction of the keep function formula for the general case, described in Definition 12.1, is more complicated. Some of its recursive steps are now explained. The fundamental idea of the construction of the keep function formula is that the formula $F_{\not\triangleleft H_D, G_E, \not\triangleleft H_I}$ must always fulfill the statement $\neg \not\triangleleft H_I \wedge G_E \Rightarrow \neg \not\triangleleft H_D$.

Firstly, in general, the TRF G_E is *normalized*[16] and transformed to its disjunctive normal form:

$$dnf(\overline{G_E}) = C_1 \vee \ldots \vee C_n$$

where C_i are conjunctions, $i = 1, \ldots, n$. Therefore:

$$\begin{aligned} &\neg \not\triangleleft H_I \wedge (C_1 \vee \ldots \vee C_n) \Rightarrow \neg \not\triangleleft H_D \Leftrightarrow \\ &(\neg \not\triangleleft H_I \wedge C_1 \Rightarrow \neg \not\triangleleft H_D) \wedge \ldots \wedge (\neg \not\triangleleft H_I \wedge C_n \Rightarrow \neg \not\triangleleft H_D) \end{aligned}$$

$F_{\not\triangleleft H_D, (C_1 \vee \ldots \vee C_n), \not\triangleleft H_I}$ must fulfill the implication $\neg \not\triangleleft H_I \wedge (C_1 \vee \ldots \vee C_n) \Rightarrow \neg \not\triangleleft H_D$ and $F_{\not\triangleleft H_D, C_i, \not\triangleleft H_I}$ must fulfill the implication $\neg \not\triangleleft H_I \wedge C_i \Rightarrow \neg \not\triangleleft H_D$ for each $i = 1, \ldots, n$. If

$$F_{\not\triangleleft H_D, (C_1 \vee \ldots \vee C_n), \not\triangleleft H_I} = F_{\not\triangleleft H_D, C_1, \not\triangleleft H_I} \wedge \ldots \wedge F_{\not\triangleleft H_D, C_n, \not\triangleleft H_I}$$

[16] [SB13], Defiition 4

then the equivalence holds:

$$(F_{\not\triangleleft H_D,(C_1\vee\ldots\vee C_n),\not\triangleleft H_I} \Rightarrow (\neg\not\triangleleft H_I \wedge G_E \Rightarrow \neg\not\triangleleft H_D)) \Leftrightarrow$$
$$(F_{\not\triangleleft H_D,C_1,\not\triangleleft H_I} \wedge \ldots \wedge F_{\not\triangleleft H_D,C_n,\not\triangleleft H_I} \Rightarrow (\neg\not\triangleleft H_I \wedge C_1 \Rightarrow \neg\not\triangleleft H_D) \wedge \ldots \wedge (\neg\not\triangleleft H_I \wedge C_n \Rightarrow \neg\not\triangleleft H_D))$$

Secondly, if $\not\triangleleft H_I = \not\triangleleft H_{I_1} \wedge \ldots \wedge \not\triangleleft H_{I_k}$, then:

$$\begin{aligned}
\neg(\not\triangleleft H_{I_1} \wedge \ldots \wedge \not\triangleleft H_{I_k}) \wedge C_E \Rightarrow \neg\not\triangleleft H_D \Leftrightarrow \\
(\neg\not\triangleleft H_{I_1} \vee \ldots \vee \neg\not\triangleleft H_{I_k}) \wedge C_E \Rightarrow \neg\not\triangleleft H_D \Leftrightarrow \\
(\neg\not\triangleleft H_{I_1} \wedge C_E) \vee \ldots \vee (\neg\not\triangleleft H_{I_n} \wedge C_E) \Rightarrow \neg\not\triangleleft H_D \Leftrightarrow \\
(\neg\not\triangleleft H_{I_1} \wedge C_E \Rightarrow \neg\not\triangleleft H_D) \wedge \ldots \wedge (\neg\not\triangleleft H_{I_n} \wedge C_E \Rightarrow \neg\not\triangleleft H_D)
\end{aligned}$$

$F_{\not\triangleleft H_D,C_E,\not\triangleleft H_{I_1}\wedge\ldots\wedge\not\triangleleft H_{I_k}}$ must fulfill the implication $\neg(\not\triangleleft H_{I_1} \wedge \ldots \wedge \not\triangleleft H_{I_k}) \wedge C_E \Rightarrow \neg\not\triangleleft H_D$ and $F_{\not\triangleleft H_D,C_E,\not\triangleleft H_{I_i}}$ must fulfill the implication $\neg\not\triangleleft H_{I_i} \wedge C_E \Rightarrow \neg\not\triangleleft H_D$ for each $i = 1,\ldots,k$, then:

$$F_{\not\triangleleft H_D,C_E,\not\triangleleft H_{I_1}\wedge\ldots\wedge\not\triangleleft H_{I_k}} = F_{\not\triangleleft H_D,C_E,\not\triangleleft H_{I_1}} \wedge \ldots \wedge F_{\not\triangleleft H_D,C_E,\not\triangleleft H_{I_n}}$$

Thirdly, if $\not\triangleleft H_D = \not\triangleleft H_{D_1} \wedge \ldots \wedge \not\triangleleft H_{D_l}$, then:

$$\begin{aligned}
\neg\not\triangleleft H_I \wedge C_E \Rightarrow \neg(\not\triangleleft H_{D_1} \wedge \ldots \wedge \not\triangleleft H_{D_l}) \quad &\Leftrightarrow \\
\neg\not\triangleleft H_I \wedge C_E \Rightarrow \neg\not\triangleleft H_{D_1} \vee \ldots \vee \neg\not\triangleleft H_{D_l} \quad &\Leftrightarrow \\
(\neg\not\triangleleft H_I \wedge C_E \Rightarrow \neg\not\triangleleft H_{D_1}) \vee \ldots \vee (\neg\not\triangleleft H_I \wedge C_E \Rightarrow \neg\not\triangleleft H_{D_l})
\end{aligned}$$

$F_{\not\triangleleft H_{D_1}\wedge\ldots\wedge\not\triangleleft H_{D_l},C_E,\not\triangleleft H_I}$ must fulfill the implication $\neg\not\triangleleft H_I \wedge C_E \Rightarrow \neg(\not\triangleleft H_{D_1} \wedge \ldots \wedge \not\triangleleft H_{D_l})$ and $F_{\not\triangleleft H_{D_i},C_E,\not\triangleleft H_I}$ must fulfill the implication $\neg\not\triangleleft H_I \wedge C_E \Rightarrow \neg\not\triangleleft H_{D_i}$ for each $i = 1,\ldots,l$, then:

$$F_{\not\triangleleft H_{D_1}\wedge\ldots\wedge\not\triangleleft H_{D_l},C_E,\not\triangleleft H_I} = F_{\not\triangleleft H_{D_1},C_E,\not\triangleleft H_I} \vee \ldots \vee F_{\not\triangleleft H_{D_l},C_E,\not\triangleleft H_I}$$

The remaining cases of Definition 12.1 with $\not\triangleleft H_I = \not\triangleleft H_{I_1} \vee \ldots \vee \not\triangleleft H_{I_k}$ and $\not\triangleleft H_D = \not\triangleleft H_{D_1} \vee \ldots \vee \not\triangleleft H_{D_l}$ are treated analogously.

The keep function f_F is based on the keep function formula $F = F_{\not\triangleleft H_D,G_E,\not\triangleleft H_I}$ as it is described in Definition 12.2. The keep function f_F yields the maximum value of b_E that fulfills the keep function formula F. The conctruction of the function has five recursive steps.

First: If $F = \top$, i.e. $dist_{C_E}(v,p) = -\infty$ and $b_E \leq b_D - (-\infty) = b_D + \infty = +\infty$, then it either means C_E is inconsistent or C_E is irrelevant for v and $f_\top = +\infty$.

Second: If $F = \bot$, i.e. $dist_{C_E}(v,p) = \infty$ and $b_E \leq b_D - (+\infty) = -\infty$, then it means b_E and b_D are independent in C_E and $f_\bot = -\infty$.

Third: If $F = b_E \leq b_D - dist_{C_E}(v,p)$, then $f_F(b_D) = b_D - dist_{C_E}(v,p) = b_E$.

Fourth: If the keep function formula is a conjunction of several conditions, e.g. $F = (b_E \leq b_{D,1} - dist_{C_E}(v_1,p_1)) \wedge (b_E \leq b_{D,2} - dist_{C_E}(v_2,p_2))$, then it is equivalent to $b_E \leq \min\{b_{D,1} - dist_{C_E}(v_1,p_1), b_{D,2} - dist_{C_E}(v_2,p_2)\}$ and $f_F = \min\{b_{D,1} - dist_{C_E}(v_1,p_1), b_{D,2} - dist_{C_E}(v_2,p_2)\}$.

Fifth: If the keep function formula is a disjunction of several conditions, e.g. $F = (b_E \leq b_{D,1} - dist_{C_E}(v_1,p_1)) \vee (b_E \leq b_{D,2} - dist_{C_E}(v_2,p_2))$, then it is equivalent to $b_E \leq \max\{b_{D,1} - dist_{C_E}(v_1,p_1), b_{D,2} - dist_{C_E}(v_2,p_2)\}$ and $f_F = \max\{b_{D,1} - dist_{C_E}(v_1,p_1), b_{D,2} - dist_{C_E}(v_2,p_2)\}$.

Definition 12 (*Keep Functions*)**.** Let G_E be a TRF, H_I and H_D be SBFs, $b_E, b_D \in BOUND$ the stream bound identifiers, $p \in ATTR$ a attribute that is assigned to b_D and $v \in ATTR \cup VAR$ is a variable that is assigned to b_E (i.e. $bounded(p, b_D)$ occurs in H_D, $bounded(v, b_E)$ occurs in H_E).

1. The *keep function formula* for $\not\triangleq H_D$, G_E and $\not\triangleq H_I$ must fulfill the implication $\neg \not\triangleq H_I \wedge G_E \Rightarrow \neg \not\triangleq H_D$ and it is defined recursively:

$$F_{\not\triangleq H_D, G_E, \not\triangleq H_I} = \bigwedge_{C_E \in dnf(\overline{G_E})} F_{\not\triangleq H_D, C_E, \not\triangleq H_I}$$

$$F_{(p > b_D), C_E, (v > b_E)} = \begin{cases} \top & \text{if } dist_{C_E}(v, p) = -\infty \\ \bot & \text{if } dist_{C_E}(v, p) = +\infty \\ b_E \leq b_D - dist_{C_E}(v, p) & \text{else} \end{cases}$$

$$\begin{aligned} F_{\not\triangleq H_D, C_E, (\not\triangleq H_{I1} \wedge \ldots \wedge \not\triangleq H_{Ik})} &= F_{\not\triangleq H_D, C_E, \not\triangleq H_{I1}} \wedge \ldots \wedge F_{\not\triangleq H_D, C_E, \not\triangleq H_{Ik}} \\ F_{\not\triangleq H_D, C_E, (\not\triangleq H_{I1} \vee \ldots \vee \not\triangleq H_{Ik})} &= F_{\not\triangleq H_D, C_E, \not\triangleq H_{I1}} \vee \ldots \vee F_{\not\triangleq H_D, C_E, \not\triangleq H_{Ik}} \\ F_{(\not\triangleq H_{D1} \wedge \ldots \wedge \not\triangleq H_{Dl}), C_E, \not\triangleq H_I} &= F_{\not\triangleq H_{D1}, C_E, \not\triangleq H_I} \vee \ldots \vee F_{\not\triangleq H_{Dl}, C_E, \not\triangleq H_I} \\ F_{(\not\triangleq H_{D1} \vee \ldots \vee \not\triangleq H_{Dl}), C_E, \not\triangleq H_I} &= F_{\not\triangleq H_{D1}, C_E, \not\triangleq H_I} \wedge \ldots \wedge F_{\not\triangleq H_{Dl}, C_E, \not\triangleq H_I} \end{aligned}$$

2. Let $F = F_{\not\triangleq H_D, G_E, \not\triangleq H_I}$ be the keep function formula for $\not\triangleq H_D$ and, G_E, $\not\triangleq H_I = (v > b_E)$. The *keep function* f_F of F is defined recursively:

$$\begin{aligned} f_\top &= +\infty \\ f_\bot &= -\infty \\ f_{b_E \leq b_D - d} &= b_D - d \\ f_{F_1 \wedge \ldots \wedge F_k} &= \min\{f_{F_1}, \ldots, f_{F_k}\} \\ f_{F_1 \vee \ldots \vee F_k} &= \max\{f_{F_1}, \ldots, f_{F_k}\} \end{aligned}$$

   where $b_E \in BOUND$ and $d \in \overline{\mathbb{Q}}$.

3. The *keep function* for $\not\triangleq H_D$ and, G_E, $\not\triangleq H_I = (v > b_E)$ is:

$$f_{\not\triangleq H_D, G_E, (v > b_E)} = f_{F_{\not\triangleq H_D, G_E, (v > b_E)}}$$

4. 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:

$$f_{b_E, q, j} = f_{\not\triangleq H_D, G_E, (v > b_E)}$$

   where $H_E = H_{I_1} \wedge \ldots \wedge H_{I_n}$ is the SBF of the schema of the TSA expression E; $n = arity(q)$ and b_E occurs in $H_{I_j} = \xi_j(H_{D_{q,j}})$; $\xi_j : a \mapsto v$ is an injective substitution that replaces attributes $a \in \mathscr{A}(D_{q,j})$ by the variables $v \in ATTR \cup VAR$, $j = 1, \ldots, n$.

Step 3. Keep values.

The keep values are lower bounds of the relevance of the input tuples. Only input tuples with attribute values greater than the keep values can contribute to further output

tuples. The keep values replace the stream bound identifiers in a relevance formula for a temporal stream definition in order to obtain a relevance condition to the temporal stream definition.

A temporal stream definition can occur in several queries in a TSA program. Moreover, a temporal stream definition can occur in one query for several times, i.e. the TSA expression of the query has the temporal stream definition as an indexed input on several positions (Definition 10). The set of relevant tuples of an input temporal stream definition[17] on one position in the query can differ from the set of relevant tuples of the same temporal stream definition on an other position in the same query. Obviously the sets of relevant tuples of an input temporal stream definition can be different for different queries. This means there exist several sets of keep values and several relevance conditions for a temporal stream definition, both with respect to a single query as well as a whole TSA program.

In general, the relevance analysis distinguishes two main kinds of keep values and relevance conditions for a temporal stream definition:

1. Keep values with respect to an indexed input on a specific position in a query. Relevance conditions that use this kind of keep values say that only tuples from a temporal stream definition with attribute values greater than the keep values are relevant for the indexed input on the position in the query.
2. Keep values with respect to the set of queries of a TSA program. Relevance conditions that use this kind of keep values say that only tuples from a temporal stream definition with attribute values greater than the keep values are relevant for the set of queries of the TSA program.

The following example illustrates how to determine keep values of a temporal stream definition with respect to an indexed input on a specific position in a query.

Let $P = (T, Q)$ be a TSA program. Let $q = (D, E) \in Q$ be a TSA query with $arity(q) = n$. Let $H_E = H_{I_1} \wedge \ldots \wedge H_{I_n}$ be the SBF of the schema of the TSA expression E. Let D' be a temporal stream definition such that the input of q on position j is $D_{q,j} = D'$, $1 \leq j \leq n$. Let $H_{D'} = bounded(p', b')$ be the SBF of the schema of D' and $H_D = bounded(p, b_D)$ be the SBF of the schema of D. Let $bounded(v, b_E)$ occur in $H_{I_j} = \xi_j(H_{D_{q,j}}) = \xi_j(H_{D'})$, where $b_E = b'$, $v = \xi_j(p')$ and $\xi_j : a \mapsto v$ is an injective substitution that replaces attributes $a \in \mathscr{A}(D_{q,j})$ by variables $v \in ATTR \cup VAR$, $j = 1, \ldots, n$. Let the attributes p' and p be related and let the TRF G_E of q specify this relation, i.e. the temporal distance[18] $dist_{G_E}(v, p)$ between v and p with respect to G_E can be determinated such that $p \leq v + dist_{G_E}(v, p)$.

The goal is to determine the keep value for b' of D' with respect to $D_{q,j}$ at run-time $t \in \mathbb{Q}$. Definition 13.2 says that the keep values of D' remain unchanged between the ends of two incremental evaluations. The keep values are computed in the end of the ith incremental evaluation and are used in relevance conditions in the beginning of the $(i+1)$th incremental evaluation. Definition 13.1 shows how to determine the keep values with respect to the

[17] Actually, a temporal stream definition does not contain any tuples. But at runtime, each temporal stream definition corresponds to exacly one temporal stream matched by the schema of the temporal stream definition. For the sake of concision the term "tuple(s) in/from the temporal stream definition" is used instead of "tuple(s) in/from the temporal stream matched the schema of the temporal stream definition such that the temporal stream definition corresponds to the temporal stream at run-time".

[18] [SB13], Definition 5

input $D_{q,j}$ on position j of the query q at the end of the ith incremental evaluation. In the beginning of the first incremental evaluation all tuples in the temporal stream definition[17] D' are relevant for $D_{q,j}$ and the keep value $rel^0_{q,j}(b')$ for the stream bound identifier b' of the temporal stream definition D' is $-\infty$. At the end of the ith incremental evaluation ($i \geq 0$) there are some new output tuples in the temporal stream definition D and there are some input tuples in D', which cannot contribute to further output tuples at the $(i+1)$th incremental evaluation. To omit these irrelevant input tuples at the $(i+1)$th incremental evaluation, it is necessary to know the current progress for the stream bound identifier b of the output temporal stream definition D computed by the current incremental evaluation. The progress value[19] $prop^i_q(b)$ reflects the progress of the output stream with respect to the query q, i.e. all output tuples with $p \leq prop^i_q(b)$ have already been derived and there are no new possible output tuples with $p \leq prop^i_q(b)$.

The keep function for the stream bound identifier b_E and the input $D_{q,j}$ on the position j of the query q, applied to the current progress value $prop^i_q(b)$ of the output temporal stream definition D, yields the keep value $rel^i_{q,j}(b')$ of the temporal stream definition D' with respect to $D_{q,j}$. This means all tuples in D' with $p' \leq rel^i_{q,j}(b')$ could contribute only to tuples with $p \leq prop^i_q(b)$ in D:

$$rel^i_{q,j}(b') = f_{b_E,q,j}(prop^i_q(b)) = (b_D - dist_{G_E}(v,p))(prop^i_q(b)) = prop^i_q(b) - dist_{G_E}(v,p)$$

The relevance condition for the temporal stream definition D' with respect to the input $D_{q,j}$ at time $e^i_q \leq t < e^{i+1}_q$, $i \geq 0$ is:

$$p' > rel^i_{q,j}(b')$$

If the temporal stream definition D' occurs once in the TSA expression E of the query q and does not occur in an other query from the set of queries Q of the TSA program P, then the obtained relevance condition with respect to the input $D_{q,j}$ is the relevance condition with respect to Q. But if D' occurs in the query q for several times (resp. in several queries in Q), then the relevance condition with respect to Q differs from the obtained condition.

The following example illustrated in Figures 9 and 10 explains the creation of relevance condition for a temporal stream definition with respect to a set of queries.

Let $q_1, q_2 \in Q$ be two queries with one common input temporal stream definition D' such that $D_{q_1,k_1} = D_{q_2,k_2} = D'$ for the positions $1 \leq k_i \leq arity(q_i)$, $i = 1, 2$ and let no other query be in Q, which has D' as an indexed input. Let $H_{D'} = bounded(p_1, b_1) \vee bounded(p_2, b_2)$ be the SBF of the schema of D'. Let $rel^{t_1}_{q_i,k_i}(b_j)$ be keep values for b_j of D' with respect to D_{q_i,k_i} at time t_1, $i, j = 1, 2$. The relevance formula for D' is $\not\triangleleft H_{D'} = p_1 > b_1 \wedge p_2 > b_2$. The relevance condition for D' at time t_1 with respect to D_{q_i,k_i} is $p_1 > rel^{t_1}_{q_i,k_i}(b_1) \wedge p_2 > rel^{t_1}_{q_i,k_i}(b_2)$, $i = 1, 2$. The application of the relevance conditions to the temporal stream definition D' yields the relevant tuples from the temporal stream definition[17] D' at time t_1 for the input D_{q_i,k_i} of the query q_i, $i = 1, 2$. Therefore, for the input D_{q_1,k_1} of the query q_1 tuples from D' with $p_1 > rel^{t_1}_{q_1,k_1}(b_1) \wedge p_2 > rel^{t_1}_{q_1,k_1}(b_2)$ are relevant. For the input D_{q_2,k_2} of the query q_2 tuples from D' with $p_1 > rel^{t_1}_{q_2,k_2}(b_1) \wedge p_2 > rel^{t_1}_{q_2,k_2}(b_2)$ are relevant as illustrated in Figure 9. The white dotted resp. the gray areas illustrate irrelevant resp. relevant tuples.

[19] [SB13], Definition 26

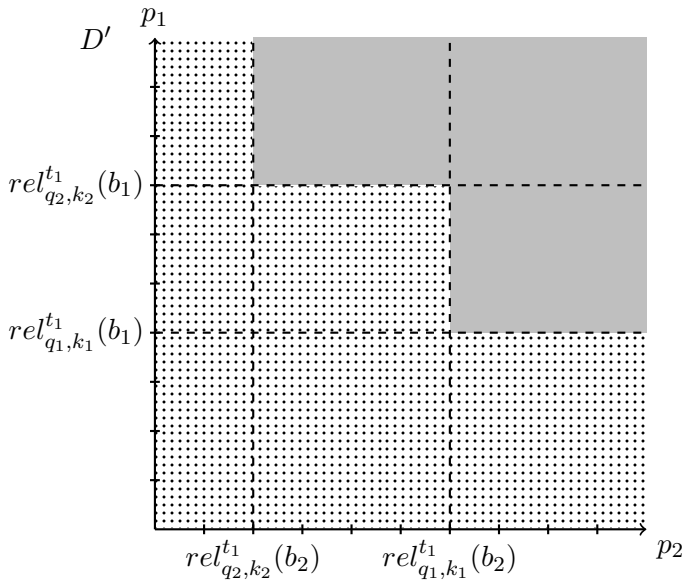

Figure 9: Relevant tuples from D' with respect to D_{q_1,k_1} and D_{q_2,k_2} at system time t_1

The determination of the relevant tuples from D' with respect to the set of queries Q is based on the following fact. Each input tuple that is relevant for at least one indexed input of a query from Q is relevant for Q. Therefore, the tuples from D', which fulfill one of the relevance conditions with respect to D_{q_i,k_i}, are relevant for Q, $i = 1, 2$:

$$(p_1 > rel^{t_1}_{q_1,k_1}(b_1) \wedge p_2 > rel^{t_1}_{q_1,k_1}(b_2)) \vee (p_1 > rel^{t_1}_{q_2,k_2}(b_1) \wedge p_2 > rel^{t_1}_{q_2,k_2}(b_2))$$

The disjunction of the relevance conditions for the temporal stream definition D' with respect to D_{q_i,k_i} determine the set of the only relevant tuples with respect to the set of queries Q, but the determination is expensive for the program because of long processing time: the sets of relevant tuples with respect to each D_{q_i,k_i} must be determined and then joined. In the example $i = 1, 2$, but in general, the number of queries that have D' as an indexed input, can be big. The following conclusion is used in order to improve the processing time of the relevance condition with respect to Q:

$$\begin{aligned}
&(p_1 > rel^{t_1}_{q_1,k_1}(b_1) \wedge p_2 > rel^{t_1}_{q_1,k_1}(b_2)) \ \vee \ (p_1 > rel^{t_1}_{q_2,k_2}(b_1) \wedge p_2 > rel^{t_1}_{q_2,k_2}(b_2)) \Leftrightarrow \\
&(p_1 > rel^{t_1}_{q_1,k_1}(b_1) \vee p_1 > rel^{t_1}_{q_2,k_2}(b_1)) \ \wedge \ (p_2 > rel^{t_1}_{q_1,k_1}(b_2) \vee p_2 > rel^{t_1}_{q_2,k_2}(b_2)) \wedge \\
&(p_1 > rel^{t_1}_{q_1,k_1}(b_1) \vee p_2 > rel^{t_1}_{q_2,k_2}(b_2)) \ \wedge \ (p_2 > rel^{t_1}_{q_1,k_1}(b_2) \vee p_1 > rel^{t_1}_{q_2,k_2}(b_1)) \Rightarrow \\
&(p_1 > rel^{t_1}_{q_1,k_1}(b_1) \vee p_1 > rel^{t_1}_{q_2,k_2}(b_1)) \ \wedge \ (p_2 > rel^{t_1}_{q_1,k_1}(b_2) \vee p_2 > rel^{t_1}_{q_2,k_2}(b_2)) \Leftrightarrow \\
&p_1 > \min\{rel^{t_1}_{q_1,k_1}(b_1), rel^{t_1}_{q_2,k_2}(b_1)\} \ \wedge \ p_2 > \min\{rel^{t_1}_{q_1,k_1}(b_2), rel^{t_1}_{q_2,k_2}(b_2)\}
\end{aligned}$$

The new condition consist of only one conjunction and the processing time of the condition for the program is less than the processing time of the disjunction of the specific relevance conditions for D'. Figure 10 illustrates the set of the relevant tuples of D' with respect to Q on the basis of the condition. Notice, that the rectangle $d_1d_2d_3d_4$ contains the irrelevant tuples for each indexed input of each query in Q at time t_1.

The values $\min\{rel^{t_1}_{q_1,k_1}(b_i), rel^{t_1}_{q_2,k_2}(b_i)\}$ are defined as the keep values for the stream bound identifier b_i with respect to the set of queries Q, $i = 1, 2$:

$$rel^{t_1}_{D'}(b_i) = \min\{rel^{t_1}_{q_1,k_1}(b_i), rel^{t_1}_{q_2,k_2}(b_i)\}$$

The formal description in Definition 13.3.

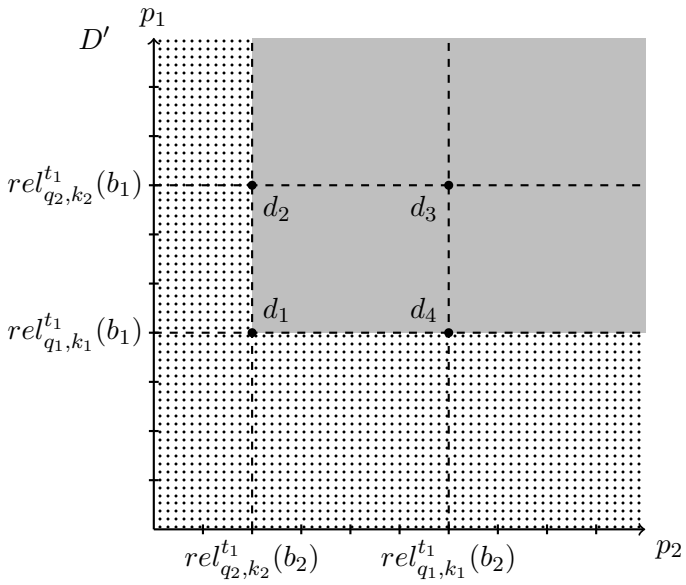

Figure 10: The relevant tuples from D' with respect to Q at time t_1

Definition 13 (*Keep Values*)**.** Let Q be a set of TSA queries. For a TSA query $q = (D, E)$ with $arity(q) = n$, let $e^i_q \in \mathbb{Q}$ denote the end time of the ith ($i \geq 1$) incremental computation for q. Let $H_E = H_{I_1} \wedge \ldots \wedge H_{I_n}$ be the SBF of the schema of the TSA expression E. Let D' be a temporal stream defnition such that the input of the query q on position j is $D_{q,j} = D'$ for $1 \leq j \leq n$. The following is defined simultaneously:

1. The *keep value* for the stream bound identifier b of the temporal stream definition D' with respect to the input $D_{q,j}$ of the query q on position j at the ith incremental computation of q, $1 \leq j \leq n$, $i \geq 0$

$$rel^i_{q,j}(b) = \begin{cases} -\infty & \text{for } i = 0 \\ f_{b,q,j}(prop^i_q(b_1), \ldots, prop^i_q(b_l)) & \text{for } i > 0 \end{cases}$$

   where b occurs in $H_{I_j} = \xi_i(H_{D_{q,j}})$; $f_{b,q,j}$ is the function term from Definition 12.4; $b_1, \ldots, b_l$ are the stream bound identifiers occurring in the SBF H_D of the schema of the output temporal stream definition D; $prop^i_q(b_j)$ are the current progress values[20], which reflects the progress of the output stream with respect to the query q; $\xi_j : a \mapsto v$ is an injective substitution, that replaces attributes $a \in \mathscr{A}(D_{q,j})$ by variables $v \in ATTR \cup VAR$, $1 \leq j \leq n$.

[20] [SB13], Definition 26

2. The *keep value* for the stream bound identifier b of the temporal stream definition D' with respect to the input $D_{q,j}$ of the query q on position j at time $t \in \mathbb{Q}$, $1 \leq j \leq n$

$$rel^t_{q,j}(b) = \begin{cases} rel^0_{q,j}(b) \text{ for } t < e^1_q \\ rel^i_{q,j}(b) \text{ for } e^i_q \leq t < e^{i+1}_q \end{cases}$$

3. The *keep value* for the stream bound identifier b of the temporal stream definition D' with respect to the set of queries Q at time $t \in \mathbb{Q}$

$$rel^t_{D'}(b) = \min_{\substack{q \in Q \\ D_{q,j} = D' \\ 1 \leq j \leq arity(q)}} \{rel^t_{q,j}(b)\}$$

Step 4. Relevance Conditions and Relevance Expressions.

There are three types of relevance conditions. Firstly (resp. secondly), it is the relevance condition for a temporal stream definition with respect to an indexed input of a TSA query at an incremental evaluation (resp. at a run-time). Thirdly, it is the relevance condition for a temporal stream definition with respect to a set of queries of a TSA program. A relevance expression for a temporal stream definition is a relevance condition applied to the temporal stream definition.

A TSA program can include static temporal streams and static queries. The static temporal streams do not progress at run-time. They already contain all tuples after the initialization of the system. The static queries are evaluated only once.

A relevance condition for a non static temporal stream definition is created from a relevance formula (Definition 11) by replacing each stream bound identifier by its keep value with respect to an indexed input at an incremental evaluation (resp. at run-time). The relevance formula for a static temporal stream definition does not contain any stream bound identifiers, because the stream bound formula for a static temporal stream definition is $\top$ and the relevance formula $\not\Delta\top = \bot$. Anyway, a static temporal stream definition can occur in a non static query (e.g. to complete the information of the output tuples). In this case, all tuples from the static temporal stream definition[17] are relevant for the non static query at an incremental evaluation (resp. at run-time).

A static query produces output tuples only once at the first incremental evaluation. This means all tuples from the input temporal streams of the static query are irrelevant for the query at each incremental evaluation except the first. A static query can contain only static input temporal stream definitions.

Definition 14 (*Relevance Conditions, Relevance Expressions*)**.** Let Q be a set of TSA queries and $q = (D, E) \in Q$ be a query with $arity(q) = n$. Let $H_E = H_{I_1} \wedge \ldots \wedge H_{I_n}$ be the SBF of the schema of the TSA expression E. Let D' be a temporal stream definition with the SBF $H_{D'} = bounded(p_1, b_1) \vee \ldots \vee bounded(p_m, b_m)$ such that the input of the query q on position j is $D_{q,j} = D'$ for $1 \leq j \leq n$. The following is defined simultaneously:

1. The *relevance condition* for the temporal stream definition D' with respect to the input $D_{q,j}$ on position j of the query q at $(i+1)$th incremental computation, $i \geq 0$,

is:

$$\not\triangleleft H^{i+1}_{q,j} = \begin{cases} \top & \text{for static } q \text{ and } i = 0 \\ \bot & \text{for static } q \text{ and } i > 0 \\ \top & \text{for non static } q \text{ and static } D' \\ p_1 > rel^i_{q,j}(b_1) \wedge \ldots \wedge p_m > rel^i_{q,j}(b_m) & \text{for non static } q \text{ and } D' \end{cases}$$

In the case of non static q and D', $\not\triangleleft H^{i+1}_{q,j}$ is relevance condition resulting from relevance formula for H_{I_j} by replacing each stream bound identifier b_k by its keep value $rel^i_{q,j}(b_k)$ of the query q at ith incremental computation.

The *relevance expression* for the relevance condition is:

$$\sigma[\not\triangleleft H^{i+1}_{q,j}](D')$$

2. The *relevance condition* and the *relevance expression* for the temporal stream definition D' with respect to the input $D_{q,j}$ on position j of the query q at time $t \in \mathbb{Q}$ is:

$$\not\triangleleft H^t_{q,j} = \begin{cases} \not\triangleleft H^1_{q,j} & \text{for } t < e^1_q \\ \not\triangleleft H^{i+1}_{q,j} & \text{for } e^i_q \leq t < e^{i+1}_q \end{cases}$$
$$\sigma[\not\triangleleft H^t_{q,j}](D')$$

3. The *relevance condition* for the temporal stream definition D' with respect to the set Q at time $t \in \mathbb{Q}$ is:

$$\not\triangleleft H^t_{D'} = \begin{cases} p_1 > rel^t_{D'}(b_1) \wedge \ldots \wedge p_m > rel^t_{D'}(b_m) & \text{for non static } D' \\ \bigvee\limits_{\substack{q \in Q \\ D_{q,j} = D' \\ 1 \leq j \leq arity(q)}} \not\triangleleft H^t_{q,j} & \text{for static } D' \end{cases}$$

The *relevance expression* for the relevance condition is:

$$\sigma[\not\triangleleft H^t_{D'}](D')$$

3.3 Garbage Collection

The garbage collection process attempts to free memory occupied by events, states and actions that are no longer in use by the program. In the case of a TSA program, the garbage collection has to delete the tuples from all input streams, which are irrelevant for the set of queries Q of the TSA program $P = (T, Q)$.

In Section 3.2 the relevance conditions are determined. They yield the relevant tuples from the temporal stream definition[17] D with respect to Q when applied to an temporal stream definition D. All other tuples from D are irrelevant for each query from Q and can be deleted. The following definition describes the *garbage collection expression*, which determines irrelevant tuples from D with respect to Q at run-time.

Definition 15 (*Garbage Collection Expression*)**.** Let Q be a set of TSA queries and D be a temporal stream definition with the SBF $H_D = bounded(p_1, b_1) \vee \ldots \vee bounded(p_m, b_m)$. The *garbage collection expression* for D with respect to Q at time $t \in \mathbb{Q}$

$$\sigma[\neg \not\triangleleft H_D^t](D) = \begin{cases} \sigma[p_1 \leq rel_D^t(b_1) \vee \ldots \vee p_m \leq rel_D^t(b_m)](D) & \text{for the non static } D' \\ \neg(\bigvee\limits_{\substack{q \in Q \\ D_{q,j} = D \\ 1 \leq j \leq arity(q)}} \not\triangleleft H_{q,j}^t) & \text{for the static } D' \end{cases}$$

where $\not\triangleleft H_D^t$ is the relevance condition for D with respect to the set of queries Q at time $t \in \mathbb{Q}$; $\not\triangleleft H_{q,j}^t$ are the relevance conditions for the temporal stream definition D with respect to the input $D_{q,j}$ on position j of the query $q \in Q$ at time $t \in \mathbb{Q}$, whereby $D_{q,j} = D$.

3.4 Query Optimization

The incremental evaluation of a query only depends on the progress values. The *increment expression*[21] applies the *incremental selection with the pass condition*[22] to the TSA expression of the query. This is done to receive new output tuples of the query at the ith incremental computation as follows. The TSA expression takes the available input tuples and computes the set of the possible output tuples. The pass condition for the ith incremental computation of the query ensures that only the output tuples arrive on the output stream, which have the attribute values less than or equal to progress values. The tuples with attribute values greater than the progress values may not arrive in the output stream at the ith incremental evaluation because they are not stable[23]. The application of the negated pass conditions for the $i-1$th incremental computation warrants that the arrived at the ith incremental evaluation output tuples are indeed new output tuples.

Anyway, the TSA expression of the query takes the whole available parts of the input streams that occur as the indexed inputs in the query. The available parts of the input streams can contain irrelevant input tuples for an indexed input of the query even after a garbage collection has been executed. This can be explained by using the example from Section 3.2 of two queries q_1 and q_2 with one common temporal stream definition D' such that $D_{q_1,k_1} = D_{q_2,k_2} = D'$ for the positions $1 \leq k_i \leq arity(q_i)$, $i = 1, 2$. Figure 10 shows the relevant tuples from the temporal stream definition[17] D' with respect to Q at time t_1. Let the garbage collection be executed at time t_1. Figure 11 shows two pictures of D' after the execution of the garbage collection, where the dotted area illustrates freed space and the non dotted area illustrates the relevant input tuples with respect to Q. The gray area of the picture on the top illustrates the relevant input tuples with respect to the input D_{q_1,k_1} on position k_1 of the query q_1. The gray area of the lower picture illustrates the relevant input tuples with respect to D_{q_2,k_2} on position k_2 of the query q_2.

Let the queries q_1 and q_2 be incrementally evaluated after the execution of the garbage collection. Let this be the i_1th incremental evaluation for the query q_1 and the i_2th for

[21] [SB13], Definition 27

[22] Pass condition for the ith incremental computation and the negated pass condition for the $i-1$th increment computation, [SB13], Definition 27

[23] [SB13], Incremental evaluation

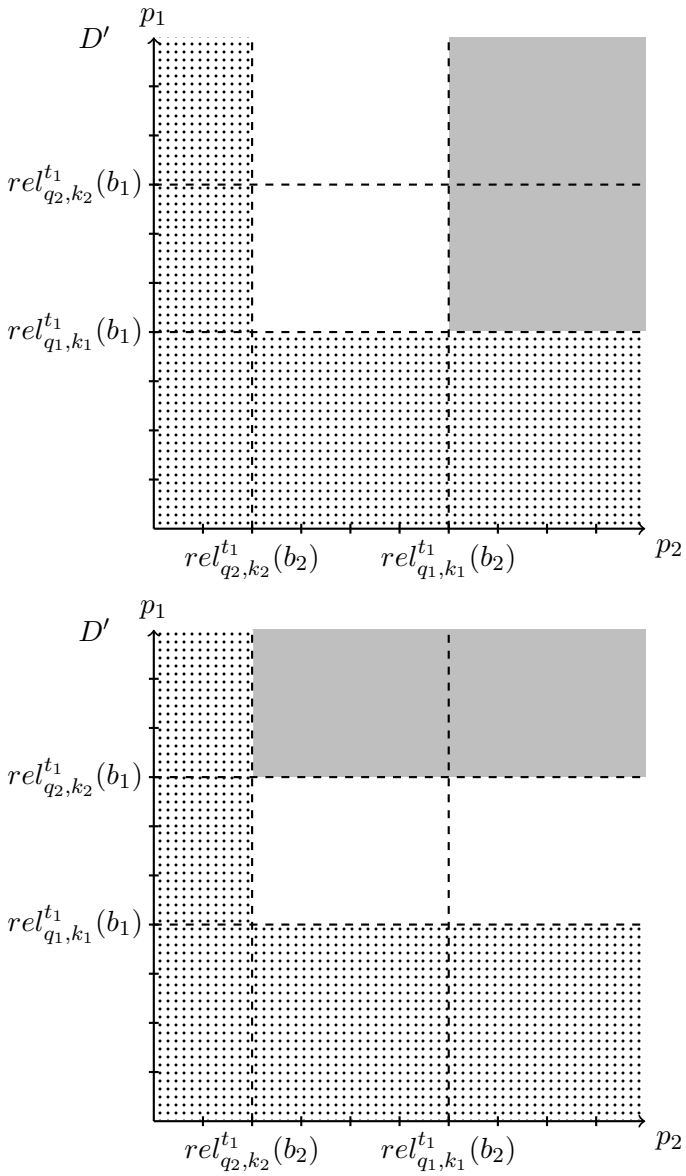

Figure 11: Stream definition D' after the execution of the garbage colection at time t_1

the qeury q_2. The complete available part of the temporal stream definition D' is taken by increment expressions of both queries for the computation of the result, although this part contains irrelevant input tuples: the white area of the picture on the top illustrates the irrelevant tuples for the input D_{q_1,k_1} of the query q_1; the white area of the lower picture illustrates the irrelevant tuples for the input D_{q_2,k_2} of the query q_2. In order to take only relevant tuples from D' and thus, to optimize the i_jth incremental evaluation of the query q_j, the relevance conditions $\not\triangleq H^{i_j}_{q_j,k_j}$ for D' with respect to the inputs D_{q_j,k_j} are applied to the temporal stream definition D' in the TSA expression of the query q_j, $j = 1, 2$. The next definition describes an optimized TSA query.

Definition 16 (*Optimized TSA query*)**.** Let $q = (D, E)$ be a TSA query. The *optimized TSA query* is the tuple $q^{opt} = (D, opt(E, 1))$, where *opt* is defined recursively:

1. $opt(D', j) = \sigma[\not\triangleq H^i_{q,j}](D')$ for a temporal stream definition D' and ith incremental evaluation.
2. $opt(op(E_1, \ldots, E_n), j) = op(opt(E_1, j + a_1), \ldots, opt(E_n, j + a_n))$ for a TSA operator *op*, where

$$a_m = \begin{cases} 0 & \text{for } m = 1 \\ \sum_{k=1}^{m-1} (arity(E_k)) & \text{for } 1 < m \leq n \end{cases}$$

Each query from a set of queries of a TSA program can be optimized with the initialization of the system immediately after the creation of non optimized queries.

4 The Relevance Analysis in Action

This section illustrates the application of the relevance analysis to the Running Example from Section 2.1. Firstly, the Dura program is translated to a TSA program. Secondly, the keep values are computed. Finally, the conditions for Garbage Collection are determined and the TSA query from the Running Example is optimized.

4.1 Translating Dura to TSA

The Dura program in Figure 1 of Section 2.1 is translated to the TSA program $P = (T, Q)$ such that:

$T = \{PreAlarm, Report, PcAlarm\}$ is the set of temporal stream definitions,

$Q = \{q_p\}$ is the set of TSA queries and $q_p = (PcAlarm, E_{PcAlarm})$,

$PreAlarm = (\text{"}PreAlarm\text{"}, S(PreAlarm))$,
$Report = (\text{"}Report\text{"}, S(Report))$,
$PcAlarm = (\text{"}PcAlarm\text{"}, S(PcAlarm))$

are temporal stream definitions with the following schemas

$\mathcal{S}(PreAlarm) = (\mathscr{A}_{PreAlarm}, G_{PreAlarm}, H_{PreAlarm})$, where

$\mathscr{A}_{PreAlarm} = \{area,\ rt.begin,\ rt.end\}$,
$G_{PreAlarm} = rt.begin \leq rt.end$,
$H_{PreAlarm} = bounded(rt.end,\ PreAlarm{:}rt.end)$

$\mathcal{S}(Report) = (\mathscr{A}_{Report},\ G_{Report},\ H_{Report})$, where
$\mathscr{A}_{Report} = \{area,\ rt.begin,\ rt.end\}$,
$G_{Report} = rt.begin \leq rt.end$,
$H_{Report} = bounded(rt.end,\ Report{:}rt.end)$

$\mathcal{S}(PcAlarm) = (\mathscr{A}_{PcAlarm},\ G_{PcAlarm},\ H_{PcAlarm})$, where
$\mathscr{A}_{PcAlarm} = \{area,\ rt.begin,\ rt.end\}$,
$G_{PcAlarm} = rt.begin \leq rt.end$,
$H_{PcAlarm} = bounded(rt.end,\ PcAlarm{:}rt.end)$

and the TSA Expression of the query q_p is

$E_{PcAlarm} = \delta[a.* \rightarrow *](\delta[* \rightarrow a.*](PreAlarm) \setminus AlarmWithReport)$, where

$$
\begin{aligned}
&AlarmWithReport = \pi[a.*](\\
&\quad \sigma[a.rt.end \leq r.rt.begin \wedge r.rt.end \leq a.rt.end + 5min](\\
&\quad\quad \sigma[a.area = r.area](\\
&\quad\quad\quad \delta[* \rightarrow a.*](PreAlarm) \times \delta[* \rightarrow r.*](Report)\\
&\quad\quad)\\
&\quad)\\
&)
\end{aligned}
$$

where $D_{q_p,1} = D_{q_p,2} = PreAlarm$ are the inputs on the positions 1 and 2; $D_{q_p,3} = Report$ is the input on the positions 3 of the query q_p.

The construction of the schema of $E_{PcAlarm}$ is divided into seven steps, where each step is an application of TSA operators. In the first five steps the schema of $AlarmWithReport$ is constructed. This intermediate result is used to construct $S(E_{PcAlarm})$. $AlarmWithReport$ contains only tuples for the pre-alarms that have been invalidated by following reports. Then for each tuple from $PreAlarm \setminus AlarmWithReport$ must be created a tuple in $PcAlarm$.

Step 1 and 2: Renaming

The construction of the schema starts with the lowest operations on input streams. In the first two steps, the attributes from the input streams are renamed as follows:

$Step_1 = \delta[* \rightarrow a.*](PreAlarm)$

$\mathcal{S}(Step_1) = (\mathscr{A}_{Step_1},\ G_{Step_1},\ H_{Step_1})$, where

$\mathscr{A}_{Step_1} = \{a.area,\ a.rt.begin,\ a.rt.end\},$
$G_{Step_1} = a.rt.begin \leq a.rt.end,$
$H_{Step_1} = bounded(a.rt.end,\ PreAlarm{:}rt.end)$

$Step_2 = \delta[* \rightarrow r.*](Report)$

$\mathcal{S}(Step_2) = (\mathscr{A}_{Step_2},\ G_{Step_2},\ H_{Step_2}),$ where
$\mathscr{A}_{Step_2} = \{r.area,\ r.rt.begin,\ r.rt.end\},$
$G_{Step_2} = r.rt.begin \leq r.rt.end,$
$H_{Step_2} = bounded(r.rt.end,\ Report{:}rt.end)$

Step 3: Cross Product

In this step, the TSA operator Cross Product is applied to the results of the previous two steps. The temporal relations between attributes from the original TRFs must hold in the resulting TRF. The resulting SBF must contain the information about attributes, which can be used to obtain a finite prefix of temporal streams. The TSA Cross Product can only be applied to streams with disjoint sets of attributes in order to avoid unintended interferences between the original TRFs and SBFs.

$Step_3 = Step_1 \times Step_2$

$\mathcal{S}(Step_3) = (\mathscr{A}_{Step_3},\ G_{Step_3},\ H_{Step_3}),$ where
$\mathscr{A}_{Step_3} = \mathscr{A}_{Step_1} \cup \mathscr{A}_{Step_2},$
$G_{Step_3} = G_{Step_1}\ \wedge\ G_{Step_2},$
$H_{Step_3} = H_{Step_1}\ \wedge\ H_{Step_2}$

Step 4: Selection Operators

This step includes the conditions from the WHERE clause and the native condition of the same area of input tuples. The part of selection conditions, which contains temporal information, is conjugated to TRF of the original expression.

$Step_4 = \sigma[a.area = r.area\ \wedge\ a.rt.end \leq r.rt.begin\ \wedge\ r.rt.end \leq a.rt.end + 5min]((Step_3))$

$\mathcal{S}(Step_4) = (\mathscr{A}_{Step_4},\ G_{Step_4},\ H_{Step_4}),$ where
$\mathscr{A}_{Step_4} = \mathscr{A}_{Step_3},$
$G_{Step_4} = a.rt.end \leq r.rt.begin\ \wedge\ r.rt.end \leq a.rt.end + 5min\ \wedge\ a.rt.begin \leq a.rt.end \wedge$
$r.rt.begin \leq r.rt.end,$
$H_{Step_4} = H_{Step_3}$

Step 5: Projection Operators

The projection is applied to the result from the fourth step. The new set of attributes contains only a part of the attributes from the schema of $Step_4$, but the temporal relations between discarded attributes must be propageted to the resulting TRF. The resulting SBF must contain the information about attributes, which can be used to obtain a finite prefix of temporal streams. For this reason, the discarded attributes are replaced by variables in the resulting TRF and SBF. The variables have the token \$.

$$Step_5 = AlarmWithReport = \pi[a.*](Step_4)$$

$$\begin{aligned}
&\mathcal{S}(Step_5) = (\mathscr{A}_{Step_5},\, G_{Step_5},\, H_{Step_5}), \text{ where}\\
&\mathscr{A}_{Step_5} = \{a.area,\, a.rt.begin,\, a.rt.end\},\\
&G_{Step_5} = a.rt.end \leq \$r.rt.begin \;\wedge\; \$r.rt.end \leq a.rt.end + 5min \;\wedge\; a.rt.begin \leq a.rt.end \;\wedge\\
&\qquad \$r.rt.begin \leq \$r.rt.end,\\
&H_{Step_5} = bounded(a.rt.end,\, PreAlarm{:}rt.end) \wedge bounded(\$r.rt.end,\, Report{:}rt.end)
\end{aligned}$$

Step 6: Set Difference Operator

The Set Difference operator is applied to two subexpressions: the first subexpression yields all pre-alarms and the second one only the pre-alarms with following reports. The information from the TRFs and the SBFs of the subexpressions is propageted to a TRF and a SBF of the resulting expression.

$$Step_6 = Step_1 \setminus Step_5$$

$$\begin{aligned}
&\mathcal{S}(Step_6) = (\mathscr{A}_{Step_6},\, G_{Step_6},\, H_{Step_6}), \text{ where}\\
&\mathscr{A}_{Step_6} = \{a.area,\, a.rt.begin,\, a.rt.end\}\\
&G_{Step_6} = G_{Step_1} \wedge (G'_1 \vee (G'_2 \;\wedge\; G'_3)\\
&G_{Step_1} = a.rt.begin \leq a.rt.end\\
&G'_1 \quad = \bigwedge_{v \in attr(H_{Step_5}) \cup vars(H_{Step_5})} ignore(\xi(v))\\
&G'_3 \quad = \xi(G_{Step_5})\\
&\qquad = \xi(a.rt.end) \leq \xi(\$r.rt.begin) \;\wedge\; \xi(\$r.rt.end) \leq \xi(a.rt.end) + 5min \;\wedge\\
&\qquad\; \xi(a.rt.begin) \leq \xi(a.rt.end) \;\wedge\; \xi(\$r.rt.begin) \leq \xi(\$r.rt.end)\\
&G'_2 \quad = (\xi(a.rt.begin) = a.rt.begin) \;\wedge\; (\xi(a.rt.end) = a.rt.end)\\
&H_{Step_6} = bounded(a.rt.end,\, PreAlarm{:}rt.end) \wedge\\
&\qquad\; bounded(\xi(a.rt.end),\, PreAlarm{:}rt.end) \wedge\\
&\qquad\; bounded(\xi(\$r.rt.end),\, Report{:}rt.end)\\
&\xi \quad = \{a.rt.* \mapsto \$2\$a.rt.*, \$r.rt.* \mapsto \$2\$r.rt.*\}
\end{aligned}$$

The atom $ignore(\xi(v))$ shows that the variable $\xi(v)$ is not a part of the namespace of the part of G_{Step_6}. The token \$2\$ shows that the variable comes from second subexpression

(from $Step_5$).

Step 7: Renaming

The goal of this step is to get the same attribute sets for $E_{PcAlarm}$ and $PcAlarm$.

$E_{PcAlarm} = \delta[a.* \rightarrow *](Step_6)$

$\mathcal{S}(E_{PcAlarm}) = (\mathscr{A}_{E_{PcAlarm}},\ G_{E_{PcAlarm}},\ H_{E_{PcAlarm}})$, where

$$
\begin{aligned}
\mathscr{A}_{E_{PcAlarm}} &= \{area,\ rt.begin,\ rt.end\} \\
G_{E_{PcAlarm}} &= G_1 \wedge (G'_1 \vee (G'_2 \ \wedge \ G'_3) \\
G_1 &= rt.begin \leq rt.end \\
G'_1 &= ignore(\$2\$a.rt.end) \ \wedge \ ignore(\$2\$r.rt.end) \\
G'_3 &= \$2\$a.rt.end \leq \$2\$r.rt.begin \ \wedge \ \$2\$r.rt.end \leq \$2\$a.rt.end + 5min \wedge \\
&\quad \$2\$a.rt.begin \leq \$2\$a.rt.end \ \wedge \ \$2\$r.rt.begin \leq \$2\$r.rt.end \\
G'_2 &= (\$2\$a.rt.begin = rt.begin) \ \wedge \ (\$2\$a.rt.end = rt.end) \\
&= (\$2\$a.rt.begin \leq rt.begin) \ \wedge \ (rt.begin \leq \$2\$a.rt.begin) \wedge \\
&\quad (\$2\$a.rt.end \leq rt.end) \ \wedge \ (rt.end \leq \$2\$a.rt.end) \\
H_{E_{PcAlarm}} &= bounded(rt.end,\ PreAlarm{:}rt.end) \wedge \\
&\quad bounded(\$2\$a.rt.end,\ PreAlarm{:}rt.end) \wedge \\
&\quad bounded(\$2\$r.rt.end,\ Report{:}rt.end)
\end{aligned}
$$

The SBF $H_{E_{PcAlarm}}$ is the conjunction $H_{I_1} \wedge H_{I_2} \wedge H_{I_3}$, where

$$
\begin{aligned}
H_{I_1} &= \xi_1(H_{D_{q_p,1}}) = \xi_1(H_{PreAlarm}),\ \xi_1 = id \\
H_{I_2} &= \xi_2(H_{D_{q_p,2}}) = \xi_2(H_{PreAlarm}),\ \xi_2 = \{a.rt.* \mapsto \$2\$a.rt.*\} \\
H_{I_3} &= \xi_3(H_{D_{q_p,3}}) = \xi_3(H_{Report}),\ \xi_3 = \{r.rt.* \mapsto \$2\$r.rt.*\}
\end{aligned}
$$

4.2 Calculation of the Keep Values

The TSA program from the Running Example is the tuple $P = (T, Q)$ with $T = \{PreAlarm,\ Report,\ PcAlarm\}$ and $Q = \{q_p\}$, where $q_p = (PcAlarm,\ E_{PcAlarm})$ is the TSA query with three inputs. The inputs on the positions 1 and 2 of the query q_p are $D_{q_p,1} = D_{q_p,2} = PreAlarm$. The input on position 3 of the query q_p is $D_{q_p,2} = Report$. Let the second incremental evaluation of the query q_p be started at $t = 14\,min$. The keep values of $PreAlarm$ and $Report$ with respect to the set of queries Q for the second incremental evaluation are computed at the end of the first incremental evaluation as follows.

As defined in Definition 13.3, the keep value for the stream bound identifier $PreAlarm{:}rt.end$ of the temporal stream definition $PreAlarm$ with respect to Q at time $t = 14\,min$ is the minimum of keep values for $PreAlarm{:}rt.end$ with respect to each input $D_{q,j}$ of each query

$q = (D, E) \in Q$ such that $D_{q,j} = PreAlarm$, $1 \leq j \leq arity(q)$:

$$rel^{14\,min}_{PreAlarm}(PreAlarm{:}rt.end) = \min_{\substack{q \in Q \\ D_{q,j}=PreAlarm \\ 1 \leq j \leq arity(q)}} \{rel^{14\,min}_{q,j}(PreAlarm{:}rt.end)\}$$
$$= \min\{rel^{14\,min}_{q_p,1}(PreAlarm{:}rt.end), rel^{14\,min}_{q_p,2}(PreAlarm{:}rt.end)\}$$

From Definition 13.2 the relevance values of $PreAlarm{:}rt.end$ with respect to $D_{q_p,1}$ and $D_{q_p,2}$ at time $t = 14\,min$ are the relevance values of $PreAlarm{:}rt.end$ with respect to $D_{q_p,1}$ and $D_{q_p,2}$ at the first incremental evaluation ($e^1_{q_p} = 12\,min \leq s^2_{q_p} = 14\,min < e^2_{q_p}$). By Definion 13.1, it is computed with the keep function for the stream bound identifier $PreAlarm{:}rt.end$ and the inputs $D_{q_p,1}$ and $D_{q_p,2}$ of the query q_p:

$$rel^1_{q_p,1}(PreAlarm{:}rt.end) = f_{PreAlarm:rt.end,q_p,1}(prop^1_{q_p}(PcAlarm{:}rt.end))$$
$$rel^1_{q_p,2}(PreAlarm{:}rt.end) = f_{PreAlarm:rt.end,q_p,2}(prop^1_{q_p}(PcAlarm{:}rt.end))$$

The progress value $prop^1_{q_p}(PcAlarm{:}rt.end)$ of the query q_p sets the boundary for the progress of the output stream on the rt axis after the first incremental evaluation. The progress value $prop^1_{q_p}(PcAlarm{:}rt.end) = 7\,min$, as shown in Figure 12.

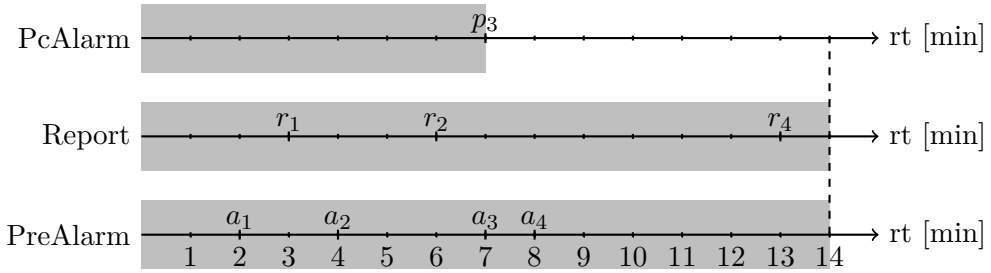

Figure 12: The progress of all streams at time $t = 14\,min$

By Definition 12.4 and 12.3, the keep functions for the stream bound identifier $PreAlarm{:}rt.end$ and the inputs $D_{q_p,1}$ and $D_{q_p,2}$ of the query q_p are:

$$f_{PreAlarm:rt.end,q_p,1}(7) = f_{\not\triangleleft H_{PcAlarm}, G_{E_{PcAlarm}}, (rt.end > PreAlarm:rt.end)}(7)$$
$$= f_{F_{\not\triangleleft H_{PcAlarm}, G_{E_{PcAlarm}}, (rt.end > PreAlarm:rt.end)}}(7)$$

$$f_{PreAlarm:rt.end,q_p,2}(7) = f_{\not\triangleleft H_{PcAlarm}, G_{E_{PcAlarm}}, (\$2\$a.rt.end > PreAlarm:rt.end)}(7)$$
$$= f_{F_{\not\triangleleft H_{PcAlarm}, G_{E_{PcAlarm}}, (\$2\$a.rt.end > PreAlarm:rt.end)}}(7)$$

Furthermore, the recursive Definition 12.1 is used for both keep function formulas from the last expressions. The TRF $G_{E_{PcAlarm}}$ is the disjunction of C_1 and C_2, therefore:

$$F_{\not\triangleleft H_{PcAlarm}, G_{E_{PcAlarm}}, (rt.end > PreAlarm:rt.end)} =$$
$$\bigwedge_{C_E \in dnf(\overline{G_E})} F_{\not\triangleleft H_{PcAlarm}, C_{E_{PcAlarm}}, (rt.end > PreAlarm:rt.end)} =$$
$$F_{\not\triangleleft H_{PcAlarm}, C_1, (rt.end > PreAlarm:rt.end)} \wedge F_{\not\triangleleft H_{PcAlarm}, C_2, (rt.end > PreAlarm:rt.end)}$$

and

$$F_{\not\triangleleft H_{PcAlarm}, G_{E_{PcAlarm}}, (\$2\$a.rt.end > PreAlarm:rt.end)} =$$
$$\bigwedge_{C_E \in dnf(\overline{G_E})} F_{\not\triangleleft H_{PcAlarm}, C_{E_{PcAlarm}}, (\$2\$a.rt.end > PreAlarm:rt.end)} =$$
$$F_{\not\triangleleft H_{PcAlarm}, C_1, (\$2\$a.rt.end > PreAlarm:rt.end)} \ \wedge \ F_{\not\triangleleft H_{PcAlarm}, C_2, (\$2\$a.rt.end > PreAlarm:rt.end)}$$

where C_1 and C_2 are the following conjunctions:

$$C_1 = \ G_1 \wedge G_1'$$
$$C_2 = \ G_1 \wedge G_2' \wedge G_3'$$

From Definition 11 the relevance formula $\not\triangleleft H_{PcAlarm}$ is:

$$\not\triangleleft H_{PcAlarm} = \ \not\triangleleft bounded(rt.end, \ PcAlarm{:}rt.end) = rt.end > PcAlarm{:}rt.end$$

Now the determination of the keep function formulas for each combination of $\not\triangleleft H_{PcAlarm}$, C_i and $(v > PreAlarm : rt.end)$ is possible, where $v \in \{rt.end, \$2\$a.rt.end\}$. From Definition 12.1 for $i = \{1, 2\}$ and for the temporal distance $d_i = dist_{C_i}(rt.end, rt.end) = 0$ follows:

$$F_{(rt.end > PcAlarm:rt.end), C_i, (rt.end > PreAlarm:rt.end)} = \begin{cases} \top, & \text{if } d_i = -\infty \\ \bot, & \text{if } d_i = +\infty \\ PreAlarm{:}rt.end \leq PcAlarm{:}rt.end - d_i, & \text{else} \end{cases}$$

For $i = \{1, 2\}$, $d_1 = dist_{C_1}(\$2\$a.rt.end, \ rt.end) = -\infty$ and $d_2 = dist_{C_2}(\$2\$a.rt.end, \ rt.end)$ $= 0$:

$$F_{(rt.end > PcAlarm:rt.end), C_i, (\$2\$a.rt.end > PreAlarm:rt.end)} = \begin{cases} \top, & \text{if } d_i = -\infty \\ \bot, & \text{if } d_i = +\infty \\ PreAlarm{:}rt.end \leq PcAlarm{:}rt.end - d_i, & \text{else} \end{cases}$$

Therefore, the keep function formulas for the whole TRF $G_{E_{PcAlarm}}$ and for each of $(v > PreAlarm : rt.end)$, where $v \in \{rt.end, \$2\$a.rt.end\}$ are:

$$F_{\not\triangleleft H_{PcAlarm}, G_{E_{PcAlarm}}, (rt.end > PreAlarm:rt.end)} = PreAlarm{:}rt.end \leq PcAlarm{:}rt.end$$

$$F_{\not\triangleleft H_{PcAlarm}, G_{E_{PcAlarm}}, (\$2\$a.rt.end > PreAlarm:rt.end)} = \top \ \wedge \ PreAlarm{:}rt.end \leq PcAlarm{:}rt.end$$
$$= PreAlarm{:}rt.end \leq PcAlarm{:}rt.end$$

and by Definition 12.2 the keep functions for the obtained keep function formulas are:

$$f_{PreAlarm:rt.end \leq PcAlarm:rt.end} = PcAlarm{:}rt.end$$

The whole calculation of the keep value for the stream bound identifier $PreAlarm{:}rt.end$

of the temporal stream definition $PreAlarm$ with respect to Q at time $t = 14\,min$ is:

$$\begin{aligned}
rel^{14\,min}_{PreAlarm}(PreAlarm{:}rt.end) &= \min\{rel^{14\,min}_{q_p,1}(PreAlarm{:}rt.end), rel^{14\,min}_{q_p,2}(PreAlarm{:}rt.end)\} \\
&= \min\{rel^{1}_{q_p,1}(PreAlarm{:}rt.end), rel^{1}_{q_p,2}(PreAlarm{:}rt.end)\} \\
&= \min\{f_{PreAlarm:rt.end,q_p,1}(prop^{1}_{q_p}(PcAlarm{:}rt.end)), \\
&\quad f_{PreAlarm:rt.end,q_p,2}(prop^{1}_{q_p}(PcAlarm{:}rt.end))\} \\
&= \min\{f_{PreAlarm:rt.end,q_p,1}(7), f_{PreAlarm:rt.end,q_p,2}(7)\} \\
&= \min\{f_{F_{\not\downarrow H_{PcAlarm},G_{E_{PcAlarm}},(rt.end>PreAlarm:rt.end)}}(7), \\
&\quad f_{F_{\not\downarrow H_{PcAlarm},G_{E_{PcAlarm}},(\$2\$a.rt.end>PreAlarm:rt.end)}}(7)\} \\
&= \min\{f_{rt.end\leq PcAlarm:rt.end}(7), f_{\$2\$a.rt.end\leq PcAlarm:rt.end}(7)\} \\
&= \min\{PcAlarm{:}rt.end(7), PcAlarm{:}rt.end(7)\} \\
&= \min\{7, 7\} \\
&= 7
\end{aligned}$$

The keep value for the stream bound identifier $Report{:}rt.end$ of the temporal stream definition $Report$ with respect to Q at time $t = 14\,min$ is computed in the same way as for $PreAlarm$:

$$\begin{aligned}
rel^{14\,min}_{Report}(Report{:}rt.end) &= \min_{\substack{q\in Q \\ D_{q,j}=Report \\ 1\leq j\leq arity(q)}} \{rel^{14\,min}_{q,j}(Report{:}rt.end)\} \\
&= rel^{14\,min}_{q_p,3}(Report{:}rt.end) \\
&= rel^{1}_{q_p,3}(Report{:}rt.end) \\
&= f_{Report:rt.end,q_p,3}(prop^{1}_{q_p}(PcAlarm{:}rt.end)) \\
&= f_{Report:rt.end,q_p,3}(7) \\
&= f_{F_{\not\downarrow H_{PcAlarm},G_{E_{PcAlarm}},(\$r.rt.end>Report:rt.end)}}(7) \\
&= f_{F_{\not\downarrow H_{PcAlarm},C_1,(\$r.rt.end>Report:rt.end)} \wedge F_{\not\downarrow H_{PcAlarm},C_2,(\$r.rt.end>Report:rt.end)}}(7) \\
&= \min\{f_{F_{\not\downarrow H_{PcAlarm},C_1,(\$r.rt.end>Report:rt.end)}}(7), \\
&\quad f_{F_{\not\downarrow H_{PcAlarm},C_2,(\$r.rt.end>Report:rt.end)}}(7)\}
\end{aligned}$$

For $i = \{1, 2\}$, $d_1 = dist_{C_1}(\$2\$r.rt.end,\, rt.end) = -\infty$ and $d_2 = dist_{C_2}(\$2\$r.rt.end,\, rt.end) = 0$

$$F_{(rt.end>PcAlarm:rt.end),C_i,(\$2\$r.rt.end>Report:rt.end)} = \begin{cases} \top, & \text{if } d_i = -\infty \\ \bot, & \text{if } d_i = +\infty \\ Report{:}rt.end \leq PcAlarm{:}rt.end - d_i, & \text{else} \end{cases}$$

Then

$$rel^{14\,min}_{Report}(Report{:}rt.end) = \min\{\infty, (PcAlarm{:}rt.end - 0)(7)\} = 7$$

4.3 Query Optimization and Garbage Collection

The keep values for the temporal stream definitions $PreAlarm$ and $Report$ with respect to the indexed inputs of the query q_p can be used to optimize this query at the ith incremental evaluation. By Definition 16, the optimized query is $q_p^{opt} = (PcAlarm, opt(E_{PcAlarm}, 1))$, where

$$\begin{aligned} opt(E_{PcAlarm}, 1) &= opt(E_1 \setminus E_2, 1) \\ &= opt(E_1, 1) \setminus opt(E_2, 2) \\ &= \delta[a.* \to *](\delta[* \to a.*](opt(PreAlarm, 1)) \setminus opt(AlarmWithReport, 2)) \end{aligned}$$

$$\begin{aligned} opt(AlarmWithReport, 2) = \pi[a.*](\\ \sigma[a.rt.end \leq r.rt.begin \wedge r.rt.end \leq a.rt.end + 5min](\\ \sigma[a.area = r.area](\\ \delta[* \to a.*](opt(PreAlarm, 2)) \times \\ \delta[* \to r.*](opt(Report, 3)) \\ ) \\ ) \\ ) \end{aligned}$$

and

$$\begin{aligned} opt(PreAlarm, 1) &= \sigma[\not\triangleleft H^i_{PcAlarm,1}](PreAlarm) \\ &= \sigma[rt.end > rel^i_{q_p,1}(PreAlarm{:}rt.end)](PreAlarm) \end{aligned}$$

$$\begin{aligned} opt(PreAlarm, 2) &= \sigma[\not\triangleleft H^i_{PcAlarm,2}](PreAlarm) \\ &= \sigma[rt.end > rel^i_{q_p,2}(PreAlarm{:}rt.end)](PreAlarm) \end{aligned}$$

$$\begin{aligned} opt(Report, 3) &= \sigma[\not\triangleleft H^i_{PcAlarm,3}](Report) \\ &= \sigma[rt.end > rel^i_{q_p,3}(Report{:}rt.end)](Report) \end{aligned}$$

The TSA expression of the optimized query q_p^{opt} only takes relevant tuples with respect to the inputs $D_{q_p,j}$, $j = 1, 2, 3$ at ith incremental evaluation. In the case of the second incremental evaluation, these are the relevant tuples from the temporal stream definitions[17] $PreAlarm$ and $Report$ with respect to the indexed inputs of the query:

$$\begin{aligned} \sigma[\not\triangleleft H^2_{PcAlarm,1}](PreAlarm)) &= \sigma[rt.end > rel^1_{q_p,1}(PreAlarm{:}rt.end)](PreAlarm) \\ &= \sigma[rt.end > 7](PreAlarm) \\ &= \{a_4\} \end{aligned}$$

$$\begin{aligned} \sigma[\not\triangleleft H^2_{PcAlarm,2}](PreAlarm)) &= \sigma[rt.end > rel^1_{q_p,2}(PreAlarm{:}rt.end)](PreAlarm) \\ &= \sigma[rt.end > 7](PreAlarm) \\ &= \{a_4\} \end{aligned}$$

$$\begin{aligned}\sigma[\not\triangleleft H^2_{PcAlarm,3}](Report)) &= \sigma[rt.end > rel^1_{q_p,3}(Report{:}rt.end)](Report)\\ &= \sigma[rt.end > 7](Report)\\ &= \{r_4\}\end{aligned}$$

Only input tuples a_4 and r_4 take part in the second incremental computation of new output tuples in the case of the optimized query q_p^{opt}. In the case of the non optimized query q_p, all available input tuples take part.

Figure 13 illustrates the relevant and irrelevant tuples for the query q_p at the second incremental evaluation from *PreAlarm* and *Report*, where the irrelevant tuples are located in the dotted area. $e^1_{q_p} = 12\,min$ is the end of the first incremental evaluation. The progress of the output temporal stream definition *PcAlarm* at time $e^1_{q_p}$ is $rt = 7\,min$. $s^2_{q_p} = 14\,min$ is the beginning of the second incremental evaluation. The input tuples located in the gray areas of the input streams are relevant for the second evaluation of *PcAlarm*.

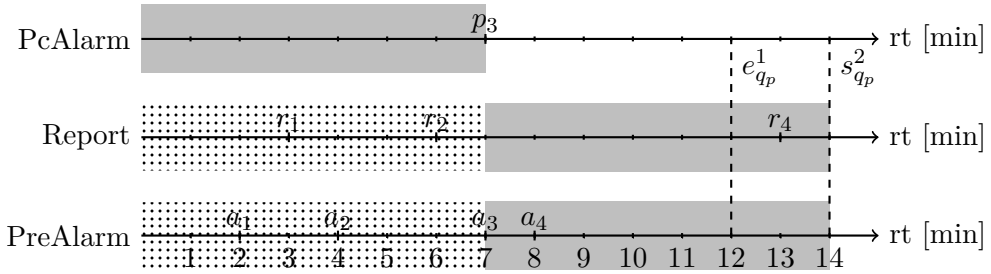

Figure 13: Relevant and irrelevant input tuples at second incremental computation

The keep values of the temporal stream definitions from the set of temporal stream definitions T remain unchanged until the end of the third incremental evaluation. The garbage collection expressions for the temporal stream definitions *PreAlarm* and *Report* with respect to the set of queries Q at time $e^2_{q_p} \leq t < e^3_{q_p}$ are:

$$\begin{aligned}\sigma[\neg\, \not\triangleleft H^t_{PreAlarm}](PreAlarm) &= \sigma[rt.end \leq rel^t_{PreAlarm}(rt.end)](PreAlarm)\\ &= \sigma[rt.end \leq 7](PreAlarm)\end{aligned}$$

$$\begin{aligned}\sigma[\neg\, \not\triangleleft H^t_{Report}](Report)) &= \sigma[rt.end \leq rel^t_{Report}(rt.end)](Report)\\ &= \sigma[rt.end \leq 7](Report)\end{aligned}$$

5 Conclusion

This bachelor thesis presented a relevance analysis for complex event queries specified in TSA. The relevance analysis allows to delete irrelevant tuples, prevents a TSA program from runnig out of memory, and improves the efficiency of the incremental evaluation..

The relevance analysis determines keep values for each data stream in a TSA program at each step of the incremental evaluation. The keep values are used to obtain relevance conditions for the data streams. The relevance conditions allow to determine those tuples

that cannot contribute to further output tuples in the following steps of the incremental evaluation.

Relevance conditions can be determined for single TSA queries and full TSA programs. The relevance conditions for single TSA queries are used to minimize the number of irrelevant tuples taking part in the evaluation of the single TSA queries. This improves the computation time of a result of the single TSA queries at each step of the incremental evaluation and thereby optimizes the queries.

The relevance conditions for a full TSA program are used to construct garbage collection. Garbage collection deletes the irrelevant tuples for a TSA program at run-time and thereby prevents the TSA program from runnig out of memory.

References

[FB12] Simon Brodt Francois Bry, Steffen Hausmann. Refinement of the implementation of event processing and eca rules for site. 2012.

[SB13] Francois Bry Simon Brodt. Analysing temporal relations – beyond windows, frames and predicates. 2013.

[SH11] Francois Bry Steffen Hausmann, Simon Brodt. Dura – concepts and examples. 2011.