Incremental Event Calculus for Run-Time Reasoning

Efthimis Tsilionis
NCSR "Demokritos", Greece
eftsilio@iit.demokritos.gr

Alexander Artikis
University of Piraeus, Greece
NCSR "Demokritos", Greece
a.artikis@unipi.gr

Georgios Paliouras
NCSR "Demokritos", Greece
paliourg@iit.demokritos.gr

ABSTRACT

We present a system for online, incremental composite event recognition. In streaming environments, the usual case is for data to arrive with a (variable) delay from, and to be retracted/revised by the underlying sources. We propose $RTEC_{inc}$, an incremental version of RTEC, a composite event recognition engine with a formal, declarative semantics, that has been shown to scale to several real-world data streams. RTEC deals with delayed arrival and retraction of events by computing at each query time composite event intervals from scratch. This often results to redundant computations. Instead, $RTEC_{inc}$ deals with delays and retractions in a more efficient way, by updating only the affected events. We evaluate $RTEC_{inc}$ theoretically, presenting a complexity analysis, and show the conditions in which it outperforms RTEC. Moreover, we compare $RTEC_{inc}$ and RTEC experimentally using two real-world datasets. The results are compatible with our theoretical analysis and show that $RTEC_{inc}$ may outperform RTEC.

CCS CONCEPTS

• **Information systems** → **Data streaming**; • **Artificial Intelligence** → *Knowledge representation and reasoning*; • **Theory of computation** → Constraint and logic programming.

KEYWORDS

incremental reasoning, stream reasoning, logic programming

ACM Reference Format:
Efthimis Tsilionis, Alexander Artikis, and Georgios Paliouras. 2019. Incremental Event Calculus for Run-Time Reasoning. In *DEBS '19: The 13th ACM International Conference on Distributed and Event-based Systems (DEBS '19), June 24–28, 2019, Darmstadt, Germany*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3328905.3329504

1 INTRODUCTION

Composite event recognition (CER) systems receive simple, derived events (SDEs) in the form of a stream in order to perform various reasoning tasks. Typically, reasoning involves the identification of composite events (CE), that is, a combination of simpler events, satisfying a given pattern. Within a CE pattern, the sub-events are subject to temporal and atemporal constraints and may be combined with (static) background knowledge. The CEs produced can then be used for real-time decision-making.

A recently proposed CER system is the Event Calculus for Run-Time Reasoning (RTEC) [6]. The Event Calculus is a logic programming formalism for representing and reasoning about events and their effects [20]. RTEC has a formal, declarative semantics, supports atemporal reasoning and reasoning over background knowledge, has built-in axioms for composite temporal phenomena, and explicitly represents CE intervals, thus avoiding the related logical problems of time-point-based event processing approaches (see [24] for a discussion).

RTEC has proven capable of real-time CER in numerous applications, such as city transport management, public space surveillance from video content [6] and maritime monitoring [2, 26]. The common case in such streaming environments is for the input events to arrive with variable delays to the CER system. Consider e.g. the maritime domain, where terrestrial and satellite stations collect position signals emitted by vessels. Some stations may have to deal with a larger amount of messages than others and as a consequence of this congestion, the events may arrive to the CER system with various delays. Moreover, revisions or retractions may be applied to input events, e.g., the start or end time of an event might have been wrong and subsequently corrected by the event source [3].

RTEC deals with delays and retractions by means of windowing; in particular, RTEC uses overlapping windows in order to 'wait' for delayed events and retractions. However, RTEC computes the intervals of CEs within a window from scratch. Thus, even if a CE remains unaffected by delays or retractions, its intervals will be recalculated, leading to inefficiency. To overcome this, we present an incremental version of RTEC, i.e. $RTEC_{inc}$, which updates only the CEs affected by delays and retractions, reduces the number of calculations and as a result improves the computational performance. The contributions of this paper are then the following:

- We present $RTEC_{inc}$, an incremental version of RTEC that avoids redundant computations by handling delays and retractions of events in a more efficient way.
- We evaluate theoretically $RTEC_{inc}$ and show the conditions in which it achieves lower computational complexity compared to RTEC.
- We compare $RTEC_{inc}$ and RTEC experimentally using real-world and synthetic datasets. The results are compatible with our complexity analysis and show that $RTEC_{inc}$ outperforms RTEC.

The structure of the paper is as follows: Section 2 summarises the functionality of RTEC. Sections 3 and 4 elaborate, respectively, on the implementation details of the incremental procedure and the complexity analysis. Section 5 presents our empirical analysis, while Section 6 discusses related work. Finally, in Section 7 we conclude with a summary and a discussion on future directions of research.

2 BACKGROUND: RUN-TIME EVENT CALCULUS

2.1 Language & Semantics

The time model of RTEC is linear and includes integer time-points. If F is a fluent — a property that is allowed to have different values at different points in time — the term $F{=}V$ denotes that fluent F has value V. $\mathsf{holdsAt}(F{=}V, T)$ is a predicate representing that fluent F has value V at time-point T. $\mathsf{holdsFor}(F{=}V, I)$ represents that I is the list of maximal intervals for which $F = V$ holds continuously. holdsAt and holdsFor are defined in such a way that, for any fluent F, $\mathsf{holdsAt}(F{=}V, T)$ if and only if T belongs to one of the maximal intervals of I for which $\mathsf{holdsFor}(F{=}V, I)$.

An *event description* in RTEC comprises rules that express: (a) event occurrences using the happensAt predicate, (b) the effects of events using the initiatedAt and terminatedAt predicates, (c) the values of fluents, with the use of the holdsAt and holdsFor predicates, as well as other, possibly atemporal, parameters. Table 1 summarises the RTEC predicates available to the event description developer. Variable names start with an upper-case letter, while predicates and constants start with a lower-case letter. Variables starting with '_' are free, i.e. they may take any value. Instantaneous simple, derived events (SDEs) and CEs are represented by means of happensAt, while CEs are represented as fluents. The majority of CEs are durative and, therefore, in CER the task generally is to compute the maximal intervals for which a fluent representing a CE has a particular value continuously. Fluents in RTEC are of two kinds: simple and statically determined. We focus on the incremental computation of the maximal intervals of simple fluents.

One of the main attractions of RTEC is that we can use the full expressivity of logic programming to represent complex temporal and atemporal constraints, as conditions in initiatedAt and terminatedAt rules for durative CEs. Next, we provide an abstract initiatedAt rule that will be our reference point. terminatedAt rules have a similar form.

$$\begin{aligned}&\mathsf{initiatedAt}(F = V, T) \leftarrow\\ &\quad \mathsf{happensAt}(A, T),\\ &\quad \mathsf{holdsAt}(B = V_B, T),\\ &\quad \mathsf{not}\ \ \mathsf{happensAt}(C, T),\\ &\quad \mathsf{not}\ \ \mathsf{holdsAt}(D = V_D, T).\end{aligned} \tag{1}$$

Rule (1) is a rule of conjunctions, in the sense that all the body literals should be satisfied in order for the rule to fire. The symbol not denotes negation by failure [11]. Variable T, present to the head and all body literals, refers to the same time-point. Rule (1) initiates $F = V$ at T if event A has occurred at T, there exists an interval of $B = V_B$ that includes T, there is no occurrence of event C at T and there is no interval of $D = V_D$ that includes T. We use the term *positive* to refer to events and fluents that occur at or include T, e.g. A and B, and the term *negative* to refer to events and fluents that do not occur at or include T (symbol not), e.g. C and D. initiatedAt and terminatedAt rules of type (1) are not restricted in the number of body literals. The only requirement is the first body literal to be a positive happensAt predicate, which can then be followed by a possibly empty set of positive/negative happensAt and holdsAt predicates.

Table 1: Main predicates of RTEC.

Predicate	Meaning
$\mathsf{happensAt}(E, T)$	Event E occurs at time T
$\mathsf{holdsAt}(F = V, T)$	The value of fluent F is V at time T
$\mathsf{holdsFor}(F = V, I)$	I is the list of maximal intervals for which $F = V$ holds continuously
$\mathsf{initiatedAt}(F = V, T)$	At time T the simple fluent $F = V$ is initiated
$\mathsf{terminatedAt}(F = V, T)$	At time T the simple fluent $F = V$ is terminated

A concrete example fluent definition, from the maritime domain [26], is presented below:

$$\begin{aligned}&\mathsf{initiatedAt}(gap(Vessel) = nearPorts, T) \leftarrow\\ &\quad \mathsf{happensAt}(gap\_start(Vessel), T),\\ &\quad \mathsf{holdsAt}(withinArea(Vessel, nearPorts) = \mathsf{true}, T).\\ &\mathsf{initiatedAt}(gap(Vessel) = farFromPorts, T) \leftarrow\\ &\quad \mathsf{happensAt}(gap\_start(Vessel), T),\\ &\quad \mathsf{not}\ \ \mathsf{holdsAt}(withinArea(Vessel, nearPorts) = \mathsf{true}, T).\\ &\mathsf{terminatedAt}(gap(Vessel) = \_Value, T) \leftarrow\\ &\quad \mathsf{happensAt}(gap\_end(Vessel), T).\end{aligned} \tag{2}$$

The above set of rules formalise the notion of a 'communication gap'. Communication gaps indicate that a vessel is not emitting its position, either due to the absence of a nearby receiving station or on purpose. Communication gaps are important in maritime monitoring since the vessel's behavior is unknown. Maritime analysts often consider communication gaps as an intention of hiding (e.g., in cases of illegal fishing in a protected area). Notice that there is a distinction between gaps occurring near ports from those occurring in the open sea. The former are usually not significant in maritime monitoring. In rule-set (2), to distinguish the two types of gaps, we represent a communication gap as a multi-valued fluent, as opposed to Boolean fluents, which are a special case where the possible values are true and false.

Rules in (2) have in their body positive events, *gap_start* and *gap_end*, and a fluent, *withinArea*(*Vessel*, *nearPorts*), that participates positively in the first rule and negatively in the second. The events *gap_start* and *gap_end* are input events produced by a module annotating vessel position streams, indicating that a vessel stopped and re-started transmitting its position [26], respectively. The fluent *withinArea*(*Vessel*, *nearPorts*) denotes the periods of time a vessel is near a port, and is produced by spatial processing of the position signals of the vessel. According to rule-set (2), a communication gap is initiated for *Vessel* if a *gap_start* has occurred near or far from ports and is terminated when a *gap_end* event is detected. Note that the terminatedAt rule in rule-set (2) terminates both types of gaps by using the free variable *_Value*.

RTEC utilises the time-points produced by initiatedAt and terminatedAt rules to construct the maximal intervals during which a fluent has a value continuously. Therefore, to compute the intervals I for which $F = V$, i.e. holdsFor($F = V, I$), we find all time-points T_s at which $F = V$ is initiated by using initiatedAt rules, and then, for each T_s, we compute the first time-point T_f after T_s at which $F = V$ is terminated by using terminatedAt rules. This is an implementation of the law of inertia.

CE definitions in RTEC are (locally) stratified logic programs [28]. Stratification allows to map all fluent-value pairs $F = V$ and all events to the non-negative integers. At level 0 we have the events and fluents that serve as input (these are the 'explicit facts' of our knowledge base). Simple fluents are output entities and thus, belong to higher strata. Events and fluents of level n are defined in terms of at least one event or fluent-value of level $n-1$ and a possibly empty set of events and fluent-values from levels lower than $n-1$. Rules in RTEC are 'safe', i.e., every variable that appears in the head of the rule or in any negative literal in the body also appears in at least one positive literal in the body.

2.2 Operation

A CER system consumes a stream of SDEs which may not necessarily be temporally ordered, in the sense that there may exist a (variable) delay between the time each SDE occurs and the time of arrival to the CER system. Furthermore, there may be revisions and retractions of SDEs. Consider for example the case where the parameters of a SDE were originally computed erroneously and subsequently revised, or the case where a SDE was reported by mistake, and the mistake was realized later [3].

In RTEC, the CER process involves the computation of the maximal intervals of fluents. This process takes place at specified query times $q_1, q_2, \ldots$. The recognition at each q_i is performed over the SDEs that fall within a specified interval, the working memory or window. All SDEs outside the window are discarded and not considered during recognition. This means that at each q_i the CER depends only on the SDEs that took place in the interval $(q_i-\omega, q_i]$. The size of ω size as well as the temporal distance between two consecutive query times — the slide step (q_i-q_{i-1}) — are user-specified.

In order to deal with delays or retractions of SDEs, the user must set ω to be longer than the slide step, i.e, $q_i-\omega < q_{i-1} < q_i$. For example, consider that an SDE occurs in the interval $(q_i-\omega, q_{i-1}]$ but arrives at RTEC only after q_{i-1}. By setting ω longer than the slide step, the effects of the SDE will be taken into account at query time q_i. Similarly, if a SDE arrived at RTEC at q_{i-1} and was subsequently retracted, the effects of retraction will be taken into consideration at q_i. However, information may still be lost. Any SDEs arriving or retracted between q_{i-1} and q_i are discarded at q_i if they took place before or at $q_i-\omega$.

At each query time q_i, RTEC computes from scratch the intervals of CEs. Figure 1 illustrates the process of computing the initiation points of $F = V$ at two consecutive query times with the use of rule (1). In this example, the window ω is longer than the step; this way, we may consider delayed arrivals or retractions of events as well as fluent intervals calculated or removed at q_i and falling in $(q_i - \omega, q_{i-1}]$. The upper part of Figure 1 displays the initiation points calculated at q_{i-1}. These points are the result of occurrences of event A and intervals of $B = V_B$ that include the occurrences of A. Additionally, event C did not occur at these time-points and the intervals of $D = V_D$ do not include them.

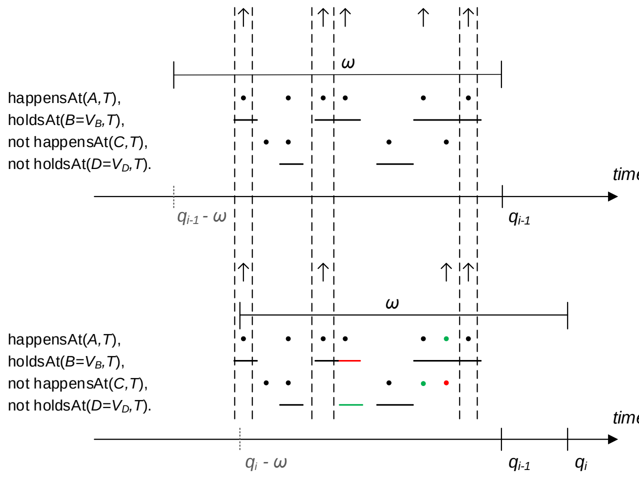


Figure 1: Initiation point computation. The upper part of the figure shows the initiation points of $F = V$, as defined by rule (1), calculated at q_{i-1}, while the bottom part shows the initiation points calculated at q_i. Dots represent event occurrences, unlabeled horizontal lines represent fluent intervals and arrows facing upwards represent initiation points. Dots coloured green express event instances that arrived at RTEC at q_i. Lines coloured green express fluent intervals that were calculated at q_i. Dots (resp. lines) coloured red express event instances (fluent intervals) that were retracted at q_i. Vertical dashed lines indicate the initiation points that are common among the two query times.

At the bottom part of Figure 1, we present the initiation points calculated at q_i. Notice the presence of delayed arrivals for events A and C (green dots), a new interval computed for $D = V_D$ (green line), a retraction of event C (red dot), and an interval reduction for $B = V_B$. The delayed arrival of event A along with the retraction of event C lead to a new initiation point. Two initiation points that were present at q_{i-1} are no longer present at q_i. The first, due to the new interval of $D = V_D$ or the diminished interval of $B = V_B$ (both have the same effect), and the second due to a delayed arrival of event C.

Three out of the four initiation points calculated at q_i are identical among the two query times. These initiation points are not affected by delays and retractions and are simply re-computed. This is clearly unnecessary and indicates the redundant computations performed by RTEC. Since RTEC employs a computation from scratch strategy, some calculations are repeated leading to inefficiency. Moreover, the same has to be done for termination points. In large event descriptions, expressing big CE hierarchies, this process can be very expensive. Thus, in the next section we present a method for incremental evaluation addressing these issues.

3 INCREMENTAL EVALUATION

We present the procedure of computing incrementally the maximal intervals of a fluent. Recall from Section 2.1 that the intervals of a fluent are produced on the basis of its initiation and termination points. Notice that some of the initiation and termination points

Algorithm 1 recogniseSimpleFluent

```
1: $\textbf{\textit{retrieve}}(F = V, se^{Q_{i-1}}, I^{Q_{i-1}})$
2: $\textit{deleteSEpoints}(F = V, se^{Q_{i-1}})$
3: $\textit{computeSEpoints}(F = V, se^{Q_{i-1}}, se^{Q_i})$
4: $\textbf{makeIntervals}(se^{Q_i}, I^{Q_i})$
5: $\textit{symmetricDifference}(I^{Q_i}, I^{Q_{i-1}}, ins, del)$
6: $\textbf{\textit{assert}}(F = V, se^{Q_i}, I^{Q_i}, ins, del)$
```

might not contribute to the intervals of the fluent at a query time q_i but might contribute to the future intervals. Assume that at the previous query time, q_{i-1}, an ending point, T_f, was calculated, but the absence of a starting point, T_s, prevented the construction of the interval $(T_s, T_f]$. Furthermore, consider that at q_i the delayed arrival of an event gives rise to the initiation point $T_s < T_f$ and as result to the interval $(T_s, T_f]$. Thus, we must store all the initiation and termination points that fall inside the overlapping part of consecutive windows, i.e. $(q_i-\omega, q_{i-1}]$.

Algorithm 1 shows the pseudo-code of recogniseSimpleFluent, the procedure for incrementally computing and storing the intervals of fluents. This procedure comprises several steps, some of which are in common with RTEC and are shown in bold in Algorithm 1. First, *RTEC_{inc}* retrieves from the computer memory the initiation and termination points $se^{Q_{i-1}}$ and the maximal intervals $I^{Q_{i-1}}$ of fluent $F = V$ computed at q_{i-1} (line 1). The intervals in $I^{Q_{i-1}}$ are temporally sorted and end in $(q_i-\omega, q_{i-1}]$.

The second step (line 2 in Algorithm 1) concerns the deletion of initiation and termination points, calculated at q_{i-1}, that no longer hold at q_i. Next, the addition phase (line 3 in Algorithm 1) consists of the calculation of new initiation and termination points, i.e. points that were not present at q_{i-1}. The details of these two steps will be presented shortly. The surviving time-points of the deletion phase and the produced time-points of the addition phase, constitute the initiation/termination points of $F = V$ at q_i, i.e. se^{Q_i}. The next step concerns the construction of the fluent intervals by means of the initiation and termination points. This process is done as in RTEC and leads to the intervals of $F = V$ at q_i, I^{Q_i} (line 4 in Algorithm 1).

RTEC$_{inc}$ computes the intervals that were added, *ins*, and retracted, *del*, for $F = V$ at q_i. The purpose of this operation lies in the fact that $F = V$ may participate, positively or negatively, in the body of initiatedAt and terminatedAt rules of a fluent of a higher stratum. This is achieved by taking the symmetric difference between the intervals of $F = V$ calculated at q_{i-1}, $I^{Q_{i-1}}$, and the intervals calculated at q_i, I^{Q_i} (line 5 in Algorithm 1). Finally, the computed list of intervals I^{Q_i}, along with the initiation/termination points calculated at q_i, se^{Q_i}, and the intervals in *ins* and *del*, are stored (line 6 in Algorithm 1), replacing the ones computed at q_{i-1}. Next, we delve into the details of the deletion and the addition phases, since these mainly differentiate *RTEC$_{inc}$* from RTEC.

3.1 Deletion phase

In the deletion phase, *RTEC$_{inc}$* examines which of the initiation and termination points falling in the overlapping part of two consecutive query times, $(q_i-\omega, q_{i-1}]$, still hold at q_i. An initiation/termination point may no longer hold for several reasons. In the case of negative events (see e.g. event C in rule (1)), the event may have occurred in the interval $(q_i-\omega, q_{i-1}]$ but arrived to the system at q_i. If the time occurrence of this delayed event coincides with an initiation/termination point calculated at q_{i-1}, the latter must be retracted. In the case of negative fluents (see fluent D in rule (1)), an interval, not present at q_{i-1}, computed at q_i and falling in $(q_i-\omega, q_{i-1}]$ may include an initiation/termination point. Again, the initiation/termination point should be removed.

In the case of positive events (see event A in rule (1)), the time of occurrence of the event may have been reported by mistake at q_{i-1}, and by q_i the mistake may have been realized and the specific event occurrence retracted. If the time of this event occurrence coincides with the time of an initiation/termination point, then this point should be deleted. Finally, in the case of positive fluents (see fluent B in rule (1)), an interval, calculated at q_{i-1} and falling in $(q_i-\omega, q_{i-1}]$, that included an initiation/termination point may be deleted or shrunk. As a consequence the interval may no longer include the initiation/termination point and once again this point should be removed.

To determine which of the initiation/termination points of $F = V$ survive, we use a transformation of the rules expressing CE definitions. Rule (3) below e.g. is a transformation of rule (1):

$$\begin{aligned} \Big[\text{initiatedAt}(F = V, T)\Big]^{se^{Q_{i-1}}} \quad &\leftarrow \\ \Big[\text{happensAt}(A, T)\Big]^{del} \quad &\vee \\ \Big[\text{holdsAt}(B = V_B, T)\Big]^{del} \quad &\vee \\ \Big[\text{happensAt}(C, T)\Big]^{ins} \quad &\vee \\ \Big[\text{holdsAt}(D = V_D, T)\Big]^{ins}. & \end{aligned} \tag{3}$$

Rule (3) is a *delta* rule determining if an initiation point must be deleted; termination points are handled in a similar manner. As opposed to rule (1), this is a rule of disjunctions. Notice that the negative predicates of rule (1) occur positively in the body of rule (3). The superscripts in the head and the body literals indicate the set in which the time argument T must belong to. Table 2 presents the definitions of these sets. We evaluate rule (3) for each initiation point of $se^{Q_{i-1}}$, i.e. for each initiation point calculated at q_{i-1} and in $(q_i-\omega, q_{i-1}]$. Since rule (3) consists of disjunctions, it suffices for the time argument of only one of the body literals to match the value of T in the head in order for the rule to be satisfied. Rule (3) fires, stating that an initiation point T in $se^{Q_{i-1}}$ should be deleted, when one of the following conditions is satisfied:

(a) T coincides with a retracted occurrence of event A (set *del*).
(b) T is included in a retracted interval of $B = V_B$ (set *del*).
(c) T matches the time-stamp of a delayed arrival of event C (set *ins*).
(d) T belongs to a new interval of $D = V_D$ calculated at q_i (set *ins*), due to a delayed event arrival or retraction.

If one of these conditions holds, the point T under investigation is removed from $se^{Q_{i-1}}$. A point survives this process, i.e., it is not removed from $se^{Q_{i-1}}$, if all conditions fail. When all points in $se^{Q_{i-1}}$ have been examined the deletion phase terminates.

Table 2: Notation of the deletion and addition phases.

Notation	Meaning
$se^{\mathrm{Q}_{i-1}}$	Initiation and termination points calculated at q_{i-1} and falling in $(q_i{-}\omega, q_{i-1}]$
del	Event occurrences and fluent intervals falling in $(q_i{-}\omega, q_{i-1}]$ and retracted at q_i
ins	Event occurrences and fluent intervals falling in $(q_i{-}\omega, q_{i-1}]$ and inserted at q_i
Q_{i-1}	Event occurrences and fluent intervals falling in $(q_i{-}\omega, q_{i-1}]$
Q_i	Event occurrences and fluent intervals falling in $(q_i - \omega, q_i]$
$\mathrm{Q}_i \backslash ins$	Event occurrences and fluent intervals at q_i without considering the ones inserted at q_i
$\mathrm{Q}_i \cup del$	Event occurrences and fluent intervals at q_i along with the ones retracted at q_i

Figure 2 illustrates the deletion phase with the use of a simple example. The evaluation of rule (1) at q_{i-1} results in the initiation points shown in the upper part of Figure 2. The occurrences of event A belonging to intervals of $B = V_B$, not coinciding with occurrences of event C and not included in the intervals of $D = V_D$, leads to the calculation of these points at q_{i-1}. In the bottom part of Figure 2, rule (3) is used to examine which of these points should be removed. The vertical dashed lines indicate the points not surviving the deletion process. The deletion of each of these points is done according to one of the four conditions outlined above. For example, the first initiation point is removed because the corresponding occurrence of A was retracted, while the last one was removed due to an interval of $D = V_D$, calculated at q_i, that includes it.

3.2 Addition phase

Once the deletion phase has completed, the addition phase commences. The addition phase consists of the calculation of new initiation and termination points, i.e. points that were not present at q_{i-1}. The new time-points may belong to the overlapping part of the two consecutive windows, $(q_i{-}\omega, q_{i-1}]$, or to the non-overlapping part, $(q_{i-1}, q_i]$. Consider rule (1) again and assume that at q_{i-1} the rule did not fire despite the fact that all body predicates except the first one satisfied the time argument T. At q_i a delayed arrival of event A with a time-stamp satisfying T will activate the rule. Similarly, deletions of event occurrences or fluent intervals may lead to the satisfaction of a rule. For example, if rule (1) did not fire at q_{i-1} due to the fact that event C occurred at T, but at q_i the specific occurrence of event C was retracted, the rule would fire.

The previous examples refer to initiation/termination points falling inside $(q_i{-}\omega, q_{i-1}]$. Initiation/termination points belonging to the non-overlapping part $(q_{i-1}, q_i]$ are not affected by delays or retractions. To calculate the new initiation points, we use the

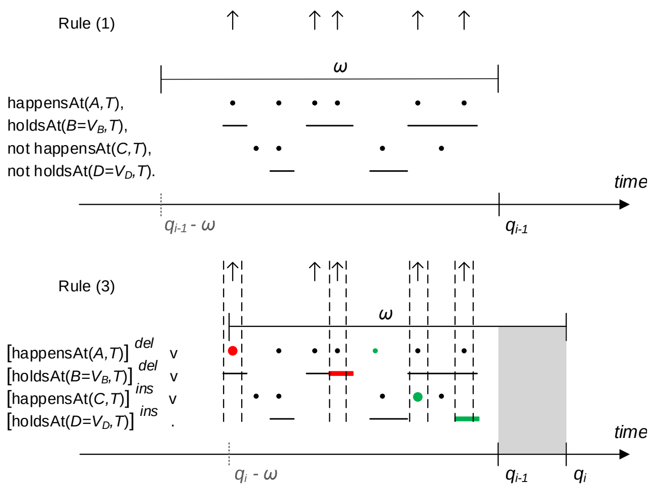


Figure 2: Illustration of the deletion phase. The upper part of the figure shows the initiation points calculated at q_{i-1} using rule (1). The bottom part presents the deletion process at q_i using the *delta* rule (3). The non-overlapping part of the two query times, $(q_{i-1}, q_i]$ is greyed out since the deletion phase concerns only the overlapping part, $(q_i{-}\omega, q_{i-1}]$. Dots represent event occurrences, unlabeled horizontal lines represent fluent intervals and arrows facing upwards represent initiation points. The black color signifies event occurrences and fluent intervals present both at q_{i-1} and q_i, the green color signifies delayed arrival of events and fluent intervals computed at q_i, while the red color signifies event occurrences and fluent intervals retracted at q_i. Enlarged dots and lines denote participation in the deletion process. The vertical dashed lines indicate the initiation points that will be removed from $se^{\mathrm{Q}_{i-1}}$.

following *delta* rules (termination points are handled similarly):

$$
\begin{array}{ll}
\begin{array}{l}
\mathsf{initiatedAt}(F = V, T) \leftarrow \\
\quad \Big[\mathsf{happensAt}(A, T)\Big]^{ins}, \\
\quad \Big[\mathsf{holdsAt}(B = V_B, T)\Big]^{\mathrm{Q}_i}, \\
\quad \mathsf{not}\ \Big[\mathsf{happensAt}(C, T)\Big]^{\mathrm{Q}_i}, \\
\quad \mathsf{not}\ \Big[\mathsf{holdsAt}(D = V_D, T)\Big]^{\mathrm{Q}_i}.
\end{array} \quad \text{(a)} &
\begin{array}{l}
\mathsf{initiatedAt}(F = V, T) \leftarrow \\
\quad \Big[\mathsf{happensAt}(A, T)\Big]^{\mathrm{Q}_i \backslash ins}, \\
\quad \Big[\mathsf{holdsAt}(B = V_B, T)\Big]^{ins}, \\
\quad \mathsf{not}\ \Big[\mathsf{happensAt}(C, T)\Big]^{\mathrm{Q}_i}, \\
\quad \mathsf{not}\ \Big[\mathsf{holdsAt}(D = V_D, T)\Big]^{\mathrm{Q}_i}.
\end{array} \quad \text{(b)} \\[2ex]
\begin{array}{l}
\mathsf{initiatedAt}(F = V, T) \leftarrow \\
\quad \Big[\mathsf{happensAt}(C, T)\Big]^{del}, \\
\quad \Big[\mathsf{happensAt}(A, T)\Big]^{\mathrm{Q}_i \backslash ins}, \\
\quad \Big[\mathsf{holdsAt}(B = V_B, T)\Big]^{\mathrm{Q}_i \backslash ins}, \\
\quad \mathsf{not}\ \Big[\mathsf{holdsAt}(D = V_D, T)\Big]^{\mathrm{Q}_i}.
\end{array} \quad \text{(c)} &
\begin{array}{l}
\mathsf{initiatedAt}(F = V, T) \leftarrow \\
\quad \Big[\mathsf{happensAt}(A, T)\Big]^{\mathrm{Q}_i \backslash ins}, \\
\quad \Big[\mathsf{holdsAt}(D = V_D, T)\Big]^{del}, \\
\quad \Big[\mathsf{holdsAt}(B = V_B, T)\Big]^{\mathrm{Q}_i \backslash ins}, \\
\quad \mathsf{not}\ \Big[\mathsf{happensAt}(C, T)\Big]^{\mathrm{Q}_i \cup del}.
\end{array} \quad \text{(d)}
\end{array}
\tag{4}
$$

The superscripts of these rules correspond to the set the time argument T is evaluated and are presented in Table 2. In rule (4)(a), event A is evaluated over the occurrences that arrived to the CER system

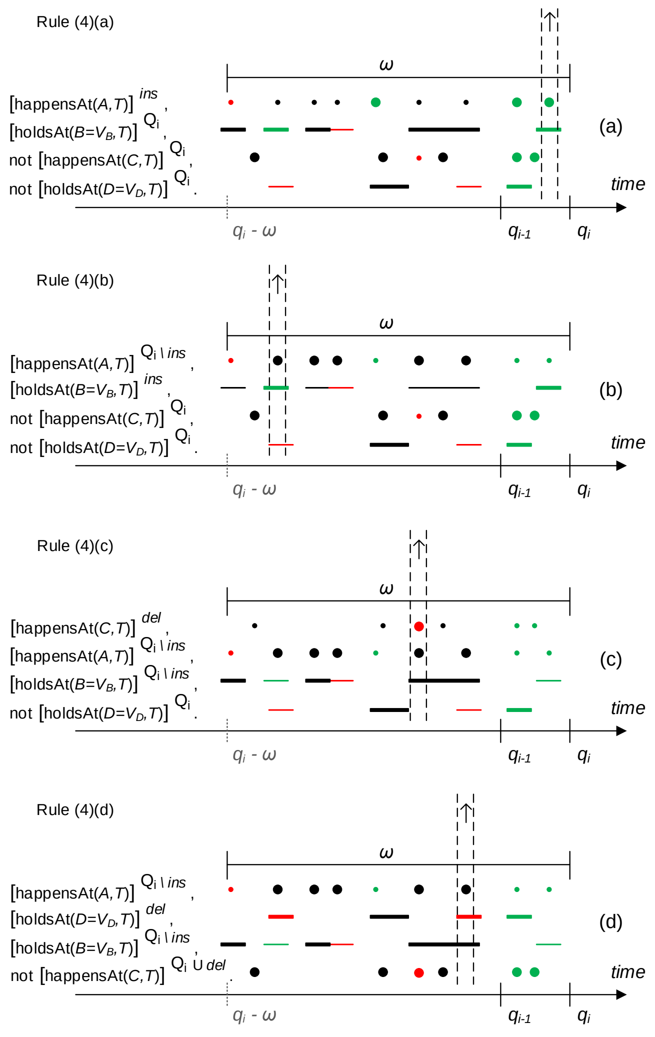

Rule (4)(a)
ω
[happensAt(A,T)]^ins ,
[holdsAt(B=V_B,T)]^Qi ,
not [happensAt(C,T)]^Qi ,
not [holdsAt(D=V_D,T)]^Qi .
(a)
time
qi - ω
qi-1
qi
Rule (4)(b)
[happensAt(A,T)]^Qi \ ins ,
[holdsAt(B=V_B,T)]^ins ,
not [happensAt(C,T)]^Qi ,
not [holdsAt(D=V_D,T)]^Qi .
(b)
Rule (4)(c)
[happensAt(C,T)]^del ,
[happensAt(A,T)]^Qi \ ins ,
[holdsAt(B=V_B,T)]^Qi \ ins ,
not [holdsAt(D=V_D,T)]^Qi .
(c)
Rule (4)(d)
[happensAt(A,T)]^Qi \ ins ,
[holdsAt(D=V_D,T)]^del ,
[holdsAt(B=V_B,T)]^Qi \ ins ,
not [happensAt(C,T)]^Qi ∪ del .
(d)

Figure 3: Illustration of the addition phase. Dots represent event occurrences, unlabeled horizontal lines represent fluents intervals and arrows facing upwards represent initiation points. The black color signifies event occurrences and fluent intervals present both at q_{i-1} and q_i, the green color signifies events arriving at, and fluent intervals computed at q_i, while the red color signifies occurrences and fluent intervals retracted by q_i. Enlarged dots and lines denote participation in the addition process. The vertical dashed lines indicate the time-points and intervals that give rise to a new initiation point. Each of the four illustrations corresponds to a *delta* rule in (4).

at q_i (set *ins*). The time-points in set *ins* are examined against all the intervals of $B = V_B$ (set Q$_i$) overlapping the interval $(q_i{-}\omega, q_i]$. If an interval of $B = V_B$ includes a time-point in set *ins*, then this time-point should not coincide with any occurrence of event C at q_i, and should not overlap any of the intervals of $D = V_D$ at q_i. If all of these conditions are satisfied, then the rule fires and gives rise to a new initiation point. Figure 3(a) shows the calculation of an initiation point belonging to the non-overlapping part $(q_{i-1}, q_i]$ by using rule (4)(a).

Rule (4)(b) is similar to (4)(a), but has a small modification which ensures that derivations are not repeated. In this rule, only the intervals computed at q_i are considered for $B = V_B$ (set *ins*). However, event A is matched against the occurrences at q_i, excluding the occurrences that were inserted to the system at q_i (set Q$_i \setminus$*ins*). This means that we examine only time-points falling inside $(q_i{-}\omega, q_{i-1}]$ and present to the system from the previous query time q_{i-1}. This is important in order to avoid repeating evaluations. If in rule (4)(b) we used all the time-points of event A at q_i, then we would have to repeat evaluations, since the inserted time-points of event A at q_i (set *ins*) are included in the occurrences of the event at q_i. The negative body literals are evaluated as in rule (4)(a). Figure 3(b) shows the calculation of an initiation point belonging to the overlapping part $(q_i{-}\omega, q_{i-1}]$ by using rule (4)(b).

Rule (4)(c) examines if a retracted occurrence of event C (set *del*) can lead to a new initiation point. Recall from rule (1) that we demand the absence of event C in order for the rule to be satisfied. Thus, if at q_i there are deleted occurrences of event C, this means that event C did not actually occur at the previously reported time-points, and thus an initiation point of $F = V$ should have been calculated. Notice that the conditions of the rule have been re-ordered for performance. Now, event C is evaluated first. This favors computation since the time-points in set *del* are usually few. Figure 3(c) shows the calculation of an initiation point belonging to the overlapping part $(q_i{-}\omega, q_{i-1}]$ by using rule (4)(c).

Rule (4)(d), examines positively the deleted intervals of $D = V_D$. At q_{i-1} an interval of $D = V_D$ may have prevented the calculation of an initiation point. If at q_i this interval of $D = V_D$ is deleted, the rule will produce a new initiation point. We employ the same optimizations as in the previous two rules, but we introduce a new one concerning negative literals. In rule (4)(c) we examined event C over the time-points in *del*. If any of these points led to new initiation points, then these derivations should not be repeated in rule (4)(d). In order to achieve this, we evaluate event C negatively over all the occurrences at q_i including the deleted ones (set Q$_i \cup$ *del*). The occurrences of event C at q_i do not include the retracted occurrences (set *del*) and by taking them into consideration we avoid repeating derivations. Figure 3(d) shows the calculation of an initiation point belonging to the overlapping part $(q_i{-}\omega, q_{i-1}]$ by using rule (4)(d).

In each of the four *delta* rules, a body literal is evaluated over the set *ins* or *del*. In practice these sets are small, compared to the set of all event occurrences and fluent intervals, i.e., set Q$_i$. By using small sets, the evaluation is faster and the performance is improved compared to the recomputation from scratch of RTEC. The introduced optimisations also ensure that a new initiation point can only be produced by one of the four *delta* rules.

4 COMPLEXITY ANALYSIS

We present a worst-case complexity analysis. We focus on the cost of computing fluent intervals in the overlapping part of two consecutive query times, $(q_i{-}\omega, q_{i-1}]$. We omit the non-overlapping part, $(q_{i-1}, q_i]$, since the cost is the same in RTEC and *RTEC$_{inc}$*. Recall

Table 3: Complexity analysis notation.

Notation	Meaning
m_{OV}	Number of time-points in $(q_i{-}\omega, q_{i-1}]$
e	Number of event types
f	Number of fluent types
t_e	Number of event occurrences in $(q_i{-}\omega, q_{i-1}]$
$\lvert I_f \rvert$	Number of fluent intervals in $(q_i{-}\omega, q_{i-1}]$
n	Number of event occurrences and fluent intervals inserted or retracted at q_i and in $(q_i{-}\omega, q_{i-1}]$
l	Number of fluent defining rules

that the time model used by RTEC is discrete. The number of time-points that are common among q_{i-1} and q_i, i.e. the time-points in $(q_i{-}\omega, q_{i-1}]$, is denoted in short m_{OV}. The maximum number of maximal intervals in m_{OV} is therefore $m_{OV}/2$. Table 3 summarises the notation of the complexity analysis. Assume that there are e events in the body of an initiatedAt/terminatedAt rule defining a fluent. e is bound in the worst case by the number of event types of the event description. Furthermore, assume that all the events have t_e occurrences in $(q_i{-}\omega, q_{i-1}]$, and additionally n delayed insertions and retractions. Similarly, assume that there are f fluents in the body of an initiatedAt/terminatedAt rule. In the worst case, this is the number of fluent types of the event description. All fluents have $|I_f|$ intervals in $(q_i{-}\omega, q_{i-1}]$, and n inserted and retracted intervals. See Table 3 for the notation.

4.1 Analysis of *RTEC_{inc}*

In the deletion phase of the incremental procedure, the initiation and termination points $se^{Q_{i-1}}$ calculated at q_{i-1} and falling in $(q_i{-}\omega, q_{i-1}]$ are examined in order to determine their validity at q_i. This is achieved by the use of *delta* rules of type (3). Assuming that there are l rules of type (1) defining a fluent, we will have l *delta* rules of type (3). In these *delta* rules each of the initiation/termination points calculated at q_{i-1} are checked against the delayed insertions of positive body events and fluents, and against the retractions of negative events and fluents. Evaluating a happensAt predicate expressing an event in the body of a *delta* rule of type(3) requires retrieving the event's inserted or retracted time-points from the computer memory, and checking whether the initiation/termination point under investigation coincides with one of the inserted/retracted time-points. Fluents are represented by means of holdsAt predicates in the body of a rule. Evaluating a holdsAt predicate requires retrieving from the computer memory the inserted or retracted intervals of the fluent, and checking whether the initiation/termination point belongs to these intervals. In the worst case, all body literals of the rules of type (3) will be evaluated, and thus, the cost of the deletion phase is bound by

$$O\Big(l \times |se^{Q_{i-1}}| \times (e + e \times n + f + f \times n)\Big) \quad , \tag{5}$$

where $|se^{Q_{i-1}}|$ represents the number of initiation/termination points calculated at q_{i-1} and falling in $(q_i{-}\omega, q_{i-1}]$.

The purpose of the addition phase is to compute the initiation/termination points that are the result of delayed insertions and retractions, by using *delta* rules of type (4). According to our initial assumptions, each initiatedAt/terminatedAt rule contains e events and f fluents. Therefore, each one of the l rules of type (1) consists of $e + f$ *delta* rules. Recall that in each of these *delta* rules, only one event or fluent is evaluated over its delayed insertions (set *ins*) or retractions (set *del*). The evaluation of the remaining events and fluents is performed as follows:

(1) If an event or fluent has not been evaluated over its insertions or retractions, it is evaluated over all its occurrences or intervals (set Q_i).
(2) If a positive event or fluent has been evaluated over its insertions, it is evaluated over all its occurrences or intervals excluding the inserted ones (set $Q_i \setminus ins$).
(3) If a negative event or fluent has been evaluated over its retractions, it is evaluated over all its occurrences or intervals, including the retracted ones (set $Q_i \cup del$).

To simplify the presentation, we assume that in the first e *delta* rules we evaluate an event over its insertions or retractions, and in the remaining f *delta* rules we evaluate a fluent over its inserted/retracted intervals. Recall that the first body literal of an initiatedAt/terminatedAt rule is restricted to be a positive happensAt predicate. Thus, in the f *delta* rules the first body literal is an event which is evaluated over all its occurrences, excluding the insertions. Moreover, in the worst case each insertion/retraction of an event and each inserted/retracted interval of a fluent will contribute a new initiation/termination point. As in the deletion phase, we have to retrieve from the computer memory the time-points and intervals according to the conditions outlined above, and perform the appropriate comparisons as indicated in each of the $e + f$ *delta* rules. The cost of computing the initiation/termination points by using all the l rules of type (4), therefore, is bound by formula (6).

In Formula (6) the first line corresponds to the cost of evaluating the e *delta* rules, and the last two lines correspond to the cost of evaluating the remaining f *delta* rules. The cost of the f *delta* rules is split in two lines to distinguish between the time-points that fail to lead to a new initiation/termination point (first line) and those that give rise to a new initiation/termination point (second line). Recall that in the f *delta* rules the first body literal is a happensAt predicate which is evaluated over all its occurrences but without considering the insertions (set $Q_i \setminus ins$). Since, in the worst case, all events and fluents contribute n initiation/termination points through their insertions/retractions, there will be some time-points that will not be promoted to new initiation/termination points. These correspond to the inserted occurrences n as well as other occurrences of the first body literal. The only time-points that will lead to a new initiation/termination point are those included in the n inserted/retracted intervals of the fluent under investigation. Therefore, the failing time-points for each f *delta* rule are $t_e - 2n$, where t_e is the number of event occurrences in $(q_i{-}\omega, q_{i-1}]$.

The final step of the incremental procedure consists of sorting the initiation/termination points, constructing intervals from these points, and calculating the inserted and retracted intervals of $F = V$ at q_i (see Algorithm 1). In the worst case, the initiation/termination points at q_i are the time-points calculated at q_{i-1} (set $se^{Q_{i-1}}$), since

$$\mathcal{O}\left(l \times \left(\begin{array}{c} (n \times e) \times \left[e + f + 1 + (e-1) \times t_e - \frac{e-1}{2} \times n + f \times |I_f|\right] + \\ (t_e - 2n) \times f \times (3+n) + \\ (n \times f) \times \left[e + f + 1 + (e-1) \times (t_e - n) + (f-1) \times |I_f| + \frac{3-f}{2} \times n\right] \end{array} \right) \right). \quad (6)$$

none of them was removed during the deletion phase, plus the time-points computed at the addition phase, $l \times n \times (e + f)$. We represent the number of initiation/termination points that hold at q_i as $|se^{Q_i}|$. The cost of sorting initiation and termination points is bound by $\mathcal{O}(|se^{Q_i}| \log |se^{Q_i}|)$ and the cost of constructing the new maximal intervals is bound by $\mathcal{O}(|se^{Q_i}|)$. Finally, the cost of the symmetric difference (see line 5 of Algorithm 1) of the intervals computed at q_{i-1}, $I^{Q_{i-1}}$, and the intervals computed at q_i, I^{Q_i}, is limited by the sum of the sizes of the two lists, as this predicate operates under the assumption that each list of maximal intervals is sorted. Since the intervals calculated at q_{i-1} are at most $|se^{Q_{i-1}}|/2$ and the intervals calculated at q_i are at most $|se^{Q_i}|/2$, the cost of the symmetric difference is bound by $\mathcal{O}(\frac{|se^{Q_i}|+|se^{Q_{i-1}}|}{2})$.

4.2 Comparison of RTEC and *RTEC$_{inc}$*

We show the conditions in which *RTEC$_{inc}$* is preferable over RTEC. The worst case scenario for RTEC does not coincide with the worst case scenario for *RTEC$_{inc}$* and vice versa. Hence, we perform two comparisons, one for the worst case of each CER engine.

4.2.1 Worst case for RTEC. In evaluation from scratch, the worst case is that every time-point m_{OV} of the overlap, $(q_i - \omega, q_{i-1}]$, is promoted to an initiation/termination point by all the initiatedAt/terminatedAt rules. Therefore, the cost of RTEC is bound by:

$$\mathcal{O}\Big(m_{OV} \times (l \times (e + f + 1 + (e-1) \times m_{OV} + f \times m_{OV}/2) + \log m_{OV} + 1) \Big). \quad (7)$$

Due to the fact that all time-points in m_{OV}, are initiation/termination points, at q_i the following hold for *RTEC$_{inc}$*:

(1) $m_{OV} = |se^{Q_i}| = t_e$.
(2) $|se^{Q_{i-1}}| = m_{OV} - n \times (e + f)$.
(3) $|I_f| = m_{OV}/2$.

Furthermore, there are no occurrences of negative events or all their occurrences are retracted at q_i, and respectively, negative fluents do not have intervals or these intervals have been removed at q_i. Concerning positive events and fluents, there are no retractions of occurrences and no retraction of intervals at q_i.

The cost of *RTEC$_{inc}$* is the sum of formulas (5), (6), and the cost formulas of sorting the initiation/termination points, constructing the intervals and taking the symmetric difference of $I^{Q_{i-1}}$ and I^{Q_i}. By using the above equalities and performing the appropriate substitutions in the total cost of *RTEC$_{inc}$* and in cost formula (7) of RTEC, we derive inequality (8), that expresses the conditions in which *RTEC$_{inc}$* is preferable to RTEC.

$$\frac{n}{m_{OV}} \times (e + f) \times \left[e \times (m_{OV} - \frac{n}{2}) + f \times \frac{m_{OV}}{2} + \frac{n}{2} - m_{OV} \times \frac{m_{OV}+n}{n} \right] + m_{OV} < 0 \quad (8)$$

Inequality (8) is simplified in order to highlight more clearly the improving conditions. According to inequality (8), the most important factor for improving the performance is the ratio $\frac{n(e+f)}{m_{OV}}$, that is, the sum of the number of delayed insertions/retractions of all events and the number of computed/retracted intervals of all fluents at q_i, to the degree of overlap.

The worst case of RTEC is an extreme case and *RTEC$_{inc}$* will usually always lead to better performance. However, the analysis of this case allows to unravel the factors that influence the most the performance of incremental evaluation. Ratio $\frac{n(e+f)}{m_{OV}}$, according to inequality (8), has the greater influence on performance. The lower this ratio is, the more probable is for *RTEC$_{inc}$* to improve the performance in terms of processing time.

4.2.2 Worst case for RTEC$_{inc}$. In incremental evaluation, the worst case is when every time-point in the overlap, m_{OV}, was an initiation/termination point at q_{i-1}, and at q_i all these points are retracted. In this case, RTEC is the preferred choice, since the cost of RTEC is close to zero, as there are no time-points in m_{OV} that can be promoted to initiation/termination points. Deleting all the initiation/termination points calculated at q_{i-1} is an inevitable operation for *RTEC$_{inc}$*. On the other hand, RTEC by computing everything from scratch avoids this operation and as a result always leads, in this case, to better performance.

5 EXPERIMENTAL RESULTS

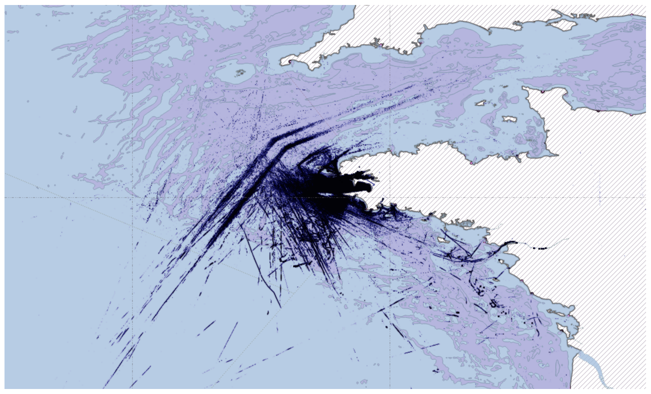
We evaluate *RTEC$_{inc}$* empirically on two real-world datasets from the field of maritime monitoring.

5.1 Experimental Setup

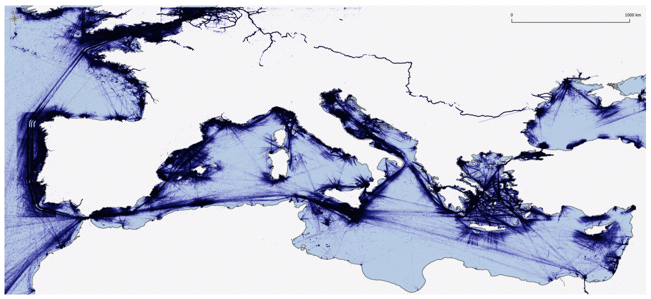
Vessels sailing at sea usually emit messages reporting their position, heading, speed, etc., at different points in time. These are the so-called AIS[1] (Automatic Identification System) messages. Temporally sorted AIS messages represent the trajectory of a vessel. A vessel trajectory may be compressed by retaining only a subset of the initial position signals, which allow for an accurate reconstruction. These 'summary' or 'critical' points include the start/end of low/high speed, changes in speed/heading, notifications about communication gaps, turns, etc. [26]. Additionally, the critical points may be spatially processed to determine whether a vessel enters or leaves an area of interest, such as a protected (Natura) area, and whether two vessels are close to each other [31]. The critical points along with the spatial relations constitute the input (SDEs) to our system. Recall, for example, the initiatedAt and terminatedAt rules presented in rule-set (2), that are used for the calculation of the maximal intervals during which a vessel turns off its AIS equipment. The events *gap_start* and *gap_end* are critical points. *withinArea*(*Vessel*, *nearPorts*) = true is a durative SDE expressing a spatial relation denoting the time periods in which a vessel is within a port area.

AIS messages are received by terrestrial or satellite stations before being forwarded to the CER system. The messages may not arrive to the CER system on time due to high traffic on the stations. These delays can be significant and the CER system must deal with them appropriately. Recall that a fluent interval can be

[1]http://www.imo.org/en/OurWork/Safety/Navigation/Pages/AIS.aspx



(a) Brest, France: Synthetic delays dataset



(b) Europe: Natural delays dataset

Figure 4: Position signals of the two datasets.

asserted/retracted due to delayed SDE arrival. In the analysis that follows, however, we do not consider the case of SDE retraction.

The two datasets used in our empirical analysis are temporally sorted. Fortunately, one of the datasets accompanies each AIS message with a time-stamp expressing the time that the message left the station. By using these timestamps it is possible to retain the natural delays. The other dataset does not provide this information and the empirical evaluation was achieved by imposing synthetic delays on the data. The geographical coverage of the datasets is shown in Figure 4.

The experiments were designed with the goal of highlighting the gain in performance $RTEC_{inc}$ brings in contrast to RTEC. The tested event description includes 16 fluents expressing various types of maritime activity, such as the movement of a vessel at a low speed, anchoring, drifting, trawling, etc [27]. The experiments were performed on a computer with 8 cores (Intel(R) Core(TM) i7-7700 CPU @ 3.6GHz) and 16 GB of RAM, running Ubuntu 16.04 LTS 64-bit with Linux Kernel 4.8.0-53-generic and YAP Prolog 6.2.2.

5.2 Synthetic delays

The first dataset is publicly available and concerns approximately $5K$ vessels sailing in the Atlantic Ocean around the port of Brest, France, and spans a period from 1 October 2015 to 31 March 2016 (see Figure 4(a)) [30]. The stream consists of approximately $5M$ input SDEs, i.e. critical and spatio-temporal events. The delays in this dataset cannot be recovered and an approach of artificially injecting delays to the stream was adopted. We performed 5 experiments, each time varying the amount of SDEs being delayed. We selected uniformly 5%, 10%, 20%, 40% and 80% of the total events to be delayed. We used a uniform distribution for selecting events, since we assume that each event has the same probability to be delayed. In the maritime domain, delays of a few hours are more probable to happen, even though delays of over 16 hours have been observed [8, 29]. In order to mimic reality as much as possible, we used a Gamma distribution to set the extent of delay. (The Gamma distribution has shape parameter $k = 2$ and scale parameter $\theta = 2$.) Thus, a delay small in time has a higher probability to be imposed in a selected event. The average delay time in all settings is approximately 8 hours.

Figures 5(a-e), display the average recognition times for windows ranging from 1 hour to 16 hours and a slide step of 1 hour. Notice that for a different percentage of delayed events, the average number of SDEs may be different for the same window size. For example, when 5% of the SDEs are delayed, the 16-hour window includes $24K$ SDEs on average, while when 10% of the SDEs are delayed, the 16-hour window contains $23.7K$ SDEs. This is due to the fact that some of the delayed events arrive to the CER system too late to be taken into consideration in a window.

As shown in Figures 5(a-e), $RTEC_{inc}$ outperforms RTEC in all experiments. The performance improvement becomes more profound as the window size increases. The experimental results confirm our complexity analysis which showed that the most important factor is the ratio of inserted/retracted occurrences of events and fluent intervals to the size of the overlap, $\frac{n(e+f)}{m_{OV}}$. The lower this ratio is the more probable is for $RTEC_{inc}$ to exhibit performance gain. Note also the linear increase in processing time, as the size of the window increases, that $RTEC_{inc}$ achieves as opposed to RTEC. However, both RTEC and $RTEC_{inc}$, regardless the size of the window, do not exhibit significant variations in their performance as the percentage of delayed events increases. This is due to the fact that the amount of SDEs per window do not change dramatically as the percentage of delayed events increases, and therefore, changes in performance are not well profound.

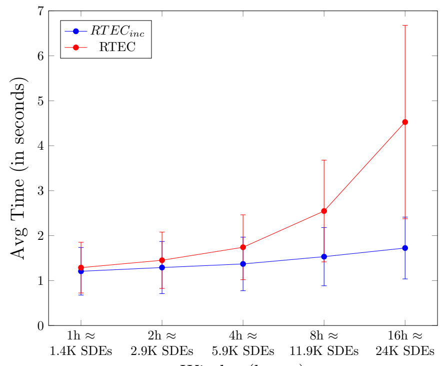
5.3 Natural delays

The second dataset concerns approximately $34K$ vessels sailing in the European seas in January 2016 and it was provided to us by IMIS Global[2], our partner in the datAcron project[3]. Figure 4(b) displays the geographical coverage of the dataset. The dataset includes $\approx 17M$ SDEs and retains the natural delays. The greatest delay observed in this dataset is approximately 14 hours. Thus, this dataset allows to test our engine in real and more demanding conditions.

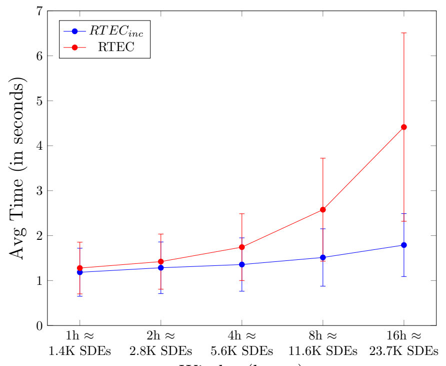
Figure 5(f) presents the experimental results. Again, we used five different temporal windows to examine the effect of the overlap on performance. The amount of SDEs is increased dramatically in each window as opposed to the Brest dataset. For example, the average number of SDEs in the 1 hour window ($\approx 37.7K$) is higher than the number of SDEs in all the 16 hours windows of Figures 5(a-e).

Similar to the previous set of experiments, $RTEC_{inc}$ outperforms RTEC. This result highlights the importance of the ratio $\frac{n(e+f)}{m_{OV}}$.

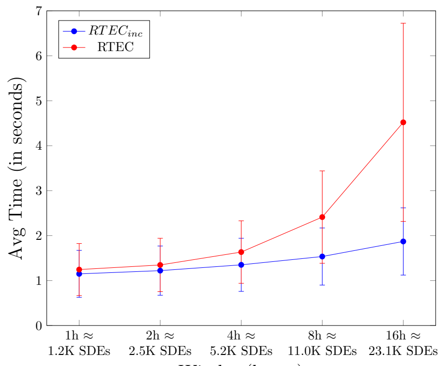
[2]https://imisglobal.com
[3]http://datacron-project.eu/



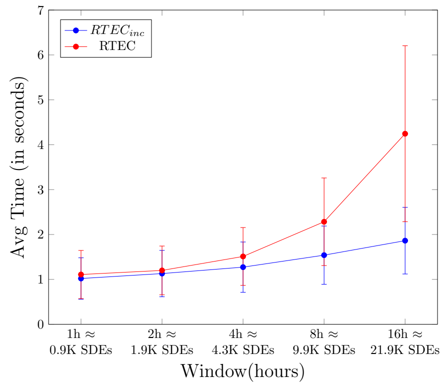
(a) Synthetic delays (5%)



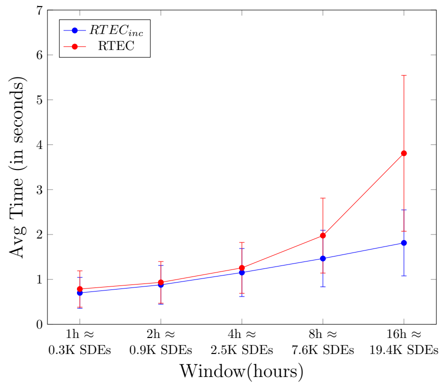
(b) Synthetic delays (10%)



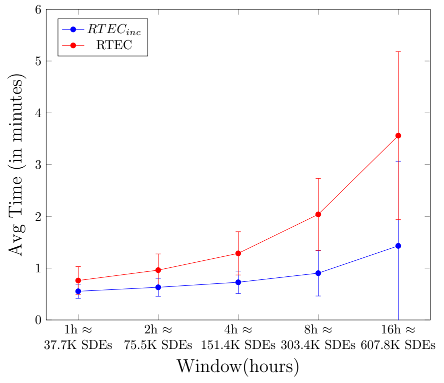
(c) Synthetic delays (20%)



(d) Synthetic delays (40%)



(e) Synthetic delays (80%)



(f) Natural delays

Figure 5: Average recognition times, with slide step of 1 hour. Diagrams (a)–(e) concern the Brest dataset, while diagram (f) concerns the dataset of Europe.

The lower this ratio is, the higher the performance gain. As the size of the window increases and as a consequence the overlap, the ratio usually becomes smaller and leads to the significant gains observed. *RTEC$_{inc}$* in contrast to RTEC, exhibits a close to linear increase in processing time as the size of the window increases.

6 RELATED WORK

RTEC has a formal, declarative semantics that support verifiable reasoning and the development of succinct CE definitions. Furthermore, by taking advantage of the power of logic programming, RTEC supports not only complex temporal constraints but also complex atemporal constraints as well as reasoning over background knowledge. Various CER systems [5, 12, 15] do not offer these types of reasoning [4]. In [25] the advantages of logic programming in CER are reported, while in [12] there is a detailed account of the limitations of existing CER systems. Another feature of RTEC is the explicit representation of CE intervals. This is in contrast to other implementations [12, 15] that suffer from the related logical problems [24].

One of the main attractions of RTEC is the ability to deal with out-of-order and/or retracted SDE streams through its windowing mechanism. The applications mentioned in [6], as well as the field of maritime monitoring [2, 26], are but a few examples in which SDEs arrive in non-chronological order and/or are retracted. The processing of events under the assumption that the stream is temporally sorted is a serious limitation of several other CER systems [12–14, 17, 22], since they cannot update, or recognise new, CEs as a result of delayed arrival or retraction of SDEs. A limitation of RTEC, however, is the computation from scratch strategy, which results, in certain circumstances, to inefficiency. In this work, we presented an incremental version of RTEC, i.e. *RTEC$_{inc}$*, that overcomes these issues. We have to mention though, that *RTEC$_{inc}$* is not always the preferable choice. In cases where there are significant updates to the recognized CEs, RTEC is the preferred option (refer to Section 4.2.2)

The presented techniques for incremental reasoning are based on the incremental maintenance of materialised views in deductive databases. A deductive database consists of base facts, called explicit facts, and a set of rules, called the program of the database. A materialised view in a deductive database is the process of computing the effects of explicit facts on rules of the program and storing the result [16]. The explicit facts can be updated through additions or deletions and thus, a new materialisation has to be produced. However, the update of explicit facts may or may not affect the inferences (implicit facts) produced after rule evaluation. The goal of incremental maintenance is, by using the updates and the state of the database before the updates, to compute the new materialisation and as a consequence to minimize the overall cost. In order for this operation to be efficient, incremental maintenance has to detect and deal only with those rule instances that require update. Notice that incremental materialisation is not always preferable over materialisation from scratch, e.g., when large amounts of explicit facts are inserted and deleted from the database [16, 23].

An update may cause a rule that fired before the update to stop firing or the opposite. The changes introduced by the updates are expressed as rules. These rules are rewritings of the original rules of the program and capture the difference between consecutive materialisations. In the literature, they are called *delta* rules and in Section 3, based on a initiatedAt rule, we provide the *delta* rules that compute the changes that have to be made in the initiation points of a fluent. Different incremental maintenance techniques have been proposed for the evaluation of the *delta* rules.

A first approach towards incremental maintenance is the counting algorithm [16]. In counting, every fact (explicit or implicit) is associated with a counter that denotes the number of times it has been derived. The counter of each fact is stored in the materialised view. When an update takes place, the counting algorithm computes the changes that have to be made in the counters of implicit facts, by using the changes made to the counters of the explicit facts as well as the previous state of the materialisation. The changes that have to be performed in the old materialisation are reflected in the counter of each fact. When a counter becomes zero, the fact must be deleted from the database. On the other hand, when the counter is positive, the fact must remain or be inserted in the view. The advantage of counting is that it only considers rules leading to the decrement or increment of a fact's counter. The limitations of the counting algorithm are the inability to handle recursive rules and the potential memory overhead caused by storing the counters [19].

The Delete/Rederive (DRed) [16] algorithm deals with recursion and does not depend on additional information in order to decide if a fact has to be removed from or be inserted in the view. DRed first 'over-deletes' all facts that depend on updated facts, that is, it evaluates all rules whose body contains an updated fact. This is called the over-deletion step. Then, it follows the re-derivation step during which the algorithm tries to find alternative derivations for each deleted fact. The body of each rule, whose head can be matched to the deleted fact, is evaluated over the new materialisation and if the evaluation succeeds the fact is inserted again to the view. The re-derivation step introduces a form of redundancy as a result of checking rule instances that fire both before and after the update. The counting algorithm avoids this inefficiency since the counter's value determines the presence of a fact in the view [19]. Finally, facts derived for the first time are inserted to the view at the insertion step.

Multiple derivations of a fact are common in RTEC, since a CE may have multiple definitions. For example, the initiation points of a fluent may depend on more than one initiatedAt rules, each of them giving rise to the same initiation point. If at query time q_i, one of these rules stops firing due to an update, DRed will delete the initiation point and then will search for other definitions that restore it. In $RTEC_{inc}$ we do not search for alternative derivations of initiation/termination points. If an initiation/termination point is retracted in the deletion phase, this may only be produced again in the addition phase. $RTEC_{inc}$ can handle multiple derivations of initiation/termination points by associating, as in the counting algorithm [16], each point with a counter that denotes the number of derivations. The CE definitions from the field of maritime monitoring used in the experimental evaluation do not result in multiple derivations of initiation/termination points, and thus, the use of a counter was omitted. However, in other applications this is not the case and the use of a counter is necessary to assure correctness.

Motik et al. [23] presented the Backward/Forward (BF) algorithm which in certain cases overcomes the redundancy introduced during the re-derivation step of DRed. If a fact is considered for deletion, BF searches for alternative derivations and only if one cannot be found it proceeds with the deletion. Hu et al. [19] proposed two hybrid approaches, the first combining DRed with counting and the second combining BF with counting.

The approaches presented so far refer to databases which are usually static. Our work transfers these ideas to a streaming environment, which is dynamic in nature and the underlying facts, i.e., SDEs, are constantly changing. The ideas of DRed and rewriting of rules have been implemented in a stream reasoning framework by Barbieri et al. [7]. These researchers associate each fact, explicit or implicit, with an expiration time according to the size of the temporal window. Then, with the use of incremental maintenance rules, they compute the facts that have to be inserted in the materialisation. However, deletion of facts depends solely on the expiration time that accompanies each fact. If a fact expires, i.e. it is no longer valid in the current temporal window, the fact is simply dropped. In our implementation, delayed arrivals and/or retractions of events as well as intervals of fluents computed and/or retracted at the current query time can also lead to retraction of initiation/termination points. Moreover, Barbieri et al. [7] do not incorporate negation in the body literals of the original rules.

An automaton-based method for out-of-order streams, called AFA (Augmented Finite Automaton) operator, achieving high throughput has been proposed in [9]. AFA produces speculative results and corrects them at the arrival of out-of-order events. The evaluation concerns one pattern and it is not clear how the proposed operator will behave in a multi-pattern context, where the patterns form hierarchies. In contrast, our solution performs multi-pattern matching in out-of-order streams and supports CE dependencies. Windowing techniques to deal with out-of-order streams have been employed in [33] with a high throughput as well as in [32], where the algorithm achieves a time complexity better than $O(\log n)$. Nevertheless, these approaches are not closely related to ours since they focus on aggregation operations.

A work that is closely related to ours, is the Cached Event Calculus (CEC) [10]. In CEC, when the intervals of a fluent are modified, the changes are propagated to fluents that rely on it. Consider for example that an event modifies the intervals of a fluent. This change is propagated to higher order fluents that depend on it. Then, another event cancels out the previous modification and the fluent intervals return to their initial state. Again, CEC considers this as a change and propagates it. The cost of this operation is very high, especially in applications where the program includes many rules with several fluents that depend on several other fluents. On the contrary, our implementation, as in [16, 19, 23], propagates changes stratum by stratum and as a consequence evaluates higher order fluents only with the necessary changes, avoiding redundant computations. If the change in a predicate is propagated immediately to predicates of higher strata, derivations and deletions of predicates can be repeated [16]. This is the limitation in the work of Küchenhoff [21].

7 SUMMARY AND FUTURE WORK

We presented an incremental version of the Event Calculus for Run-Time reasoning ($RTEC_{inc}$), a formal computational framework for composite event recognition which deals efficiently with the delayed arrival and retraction of events. $RTEC_{inc}$ avoids unnecessary calculations and improves performance, by using techniques reminiscent of the incremental maintenance of deductive databases. Our empirical evaluation on two large, real-word datasets confirmed our theoretical analysis, and illustrated the conditions in which $RTEC_{inc}$ is preferable. Besides performing a series of optimizations that can boost performance, we intend to extend our method in order to handle recursive definitions. Moreover, we aim to compare our techniques against other approaches, such as those based on automata [1, 18].

ACKNOWLEDGMENTS

This work was funded partly by the EU H2020 datACRON "Big Data Analytics for Time Critical Mobility Forecasting" project and partly by the EU H2020 INFORE "Interactive Extreme-Scale Analytics and Forecasting" project.

REFERENCES

[1] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. 2008. Efficient Pattern Matching over Event Streams. In *2008 ACM SIGMOD*. 147–160.
[2] E. Alevizos, A. Artikis, K. Patroumpas, M. Vodas, Y. Theodoridis, and N. Pelekis. 2015. How not to drown in a sea of information: An event recognition approach. In *2015 IEEE International Conference on Big Data*. 984–990.
[3] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic. 2012. Real-time Complex Event Recognition and Reasoning - A Logic Programming Approach. *Applied Artificial Intelligence* 26, 1-2 (2012), 6–57.
[4] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic. 2012. Real-time complex event recognition and reasoning-a logic programming approach. *AAI* 26 (2012), 6–57.
[5] A. Arasu, S. Babu, and J. Widom. 2006. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *The VLDB Journal* 15, 2 (2006), 121–142.
[6] A. Artikis, M. J. Sergot, and G. Paliouras. 2015. An Event Calculus for Event Recognition. *IEEE Trans. Knowl. Data Eng.* 27, 4 (2015), 895–908.
[7] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. 2010. Incremental Reasoning on Streams and Rich Background Knowledge. In *The Semantic Web: Research and Applications*. Springer Berlin Heidelberg, 1–15.
[8] E. Camossi, A.-L. Jousselme, C. Ray, M. Hadzagic, R. Dreo, and C. Claramunt. 2017. Maritime Experiments Specification, H2020 datAcron project deliverable D5.3. (2017). http://datacron-project.eu/.
[9] B. Chandramouli, J. Goldstein, and D. Maier. 2010. High-performance Dynamic Pattern Matching over Disordered Streams. *Proc. VLDB Endow.* 3, 1-2, 220–231.
[10] L. Chittaro and A. Montanari. 1996. Efficient Temporal Reasoning in the Cached Event Calculus. *Computational Intelligence* 12, 3 (1996), 359–382.
[11] K. L. Clark. 1978. *Negation as Failure*. Springer US, 293–322.
[12] G. Cugola and A. Margara. 2010. TESLA: A formally defined event specification language. *DEBS 2010*, 50–61.
[13] G. Cugola and A. Margara. 2012. Complex event processing with T-REX. *Journal of Systems and Software* 85, 8 (2012), 1709 – 1728.
[14] N. Dindar, P. M. Fischer, M. Soner, and N. Tatbul. 2011. Efficiently Correlating Complex Events over Live and Archived Data Streams. In *DEBS 2011*. 243–254.
[15] C. Dousson and P. Le Maigat. 2007. Chronicle Recognition Improvement Using Temporal Focusing and Hierarchization. In *IJCAI 2007*. 324–329.
[16] A. Gupta, I.S. Mumick, and V. S. Subrahmanian. 1993. Maintaining Views Incrementally. *SIGMOD Rec.* 22, 2 (June 1993), 157–166.
[17] D. Gyllstrom, E. Wu, H.-J. Chae, Y. Diao, P.k Stahlberg, and G. Anderson. 2006. SASE: Complex Event Processing over Streams. *CoRR* abs/cs/0612128 (2006).
[18] M. Hirzel. 2012. Partition and Compose: Parallel Complex Event Processing. In *DEBS 2012*. 191–200.
[19] P. Hu, B. Motik, and I. Horrocks. 2017. Optimised Maintenance of Datalog Materialisations. *CoRR* abs/1711.03987 (2017).
[20] R. A. Kowalski and M. J. Sergot. 1986. A Logic-based Calculus of Events. *New Generation Comput.* 4, 1 (1986), 67–95.
[21] V. Küchenhoff. 1991. On the efficient computation of the difference between consecutive database states. In *Deductive and Object-Oriented Databases*. 478–502.
[22] M. Li, M. Mani, E. A. Rundensteiner, and T. Lin. 2011. Complex Event Pattern Detection over Streams with Interval-based Temporal Semantics. In *DEBS 2011*. 291–302.
[23] B. Motik, Y. Nenov, R. Piro, and I. Horrocks. 2015. Incremental Update of Datalog Materialisation: The Backward/Forward Algorithm. In *AAAI 2015*. 1560–1568.
[24] A. Paschke. 2005. *ECA-RuleML: An Approach combining ECA Rules with Temporal Interval-Based KR Event/Action Logics and Transactional Update Logics*. Technical Report. CoRR abs/cs/0610167.
[25] A. Paschke and M. Bichler. 2006. Knowledge Representation Concepts for Automated SLA Management. *Decision Support Systems* 46 (11 2006), 187–205.
[26] K. Patroumpas, E. Alevizos, A. Artikis, M. Vodas, N. Pelekis, and Y. Theodoridis. 2017. Online event recognition from moving vessel trajectories. *GeoInformatica* 21, 2 (2017), 389–427.
[27] M. Pitsikalis, A. Artikis, R. Dreo, C. Ray, E. Camossi, and A.-L. Jousselme. 2019. Composite Event Recognition for Maritime Monitoring. In *DEBS 2019*.
[28] T. Przymusinski. 1987. On the Declarate Semantics of Stratified Deductive Databases and Logic Programs. In *Foundations of Deductive Databases and Logic Programming*. Morgan.
[29] C. Ray, E. Camossi, A.-L. Jousselme, M. Hadzagic, C. Claramunt, and E. Batty. 2016. Maritime Data Preparation and Curation, H2020 datAcron project deliverable D5.2. (2016). http://datacron-project.eu/.
[30] C. Ray, R. Dréo, E. Camossi, and A.-L. Jousselme. 2018. Heterogeneous Integrated Dataset for Maritime Intelligence, Surveillance, and Reconnaissance (Version 0.1). https://zenodo.org/record/1167595.
[31] G. M. Santipantakis, A. Vlachou, C. Doulkeridis, A. Artikis, I. Kontopoulos, and G. A. Vouros. 2018. A Stream Reasoning System for Maritime Monitoring. In *TIME 2018, Warsaw, Poland*. 20:1–20:17.
[32] K. Tangwongsan, M. Hirzel, and S. Schneider. 2018. Sub-O(log n) Out-of-Order Sliding-Window Aggregation. *CoRR* abs/1810.11308 (2018).
[33] J. Traub, P. Grulich, A. R. Cuéllar, S. Breß, A. Katsifodimos, T. Rabl, and V. Markl. 2018. Scotty: Efficient Window Aggregation for out-of-order Stream Processing. In *Proceedings of the 34th IEEE International Conference on Data Engineering*.