Incremental Event Calculus for Run-Time Reasoning[*] (Extended Abstract)

Efthimis Tsilionis[1,2] , **Alexander Artikis**[3,2] and **Georgios Paliouras**[2]
[1]Department of Informatics & Telecommunications, National and Kapodistrian University of Athens, Greece
[2]Institute of Informatics & Telecommunications, NCSR "Demokritos", Greece
[3]Department of Maritime Studies, University of Piraeus, Greece
eftsilio@{di.uoa.gr, iit.demokritos.gr}, a.artikis@unipi.gr, 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 revised by, the underlying sources. We propose $RTEC_{inc}$, an incremental version of RTEC, a composite event recognition engine with formal, declarative semantics, that has been shown to scale to several real-world data streams. RTEC deals with delayed arrival and revision of events by computing all queries from scratch. This is often inefficient since it results in redundant computations. Instead, $RTEC_{inc}$ deals with delays and revisions in a more efficient way, by updating only the affected queries. 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 in many practical cases.

1 Introduction

Streaming environments combine *simple, derived events* (SDEs) in order to recognise in real-time *composite events* (CEs) that satisfy a given pattern. These patterns are collections of simpler events, which are subject to temporal and atemporal constraints, and may be combined with static background knowledge [Cugola and Margara, 2012; Alevizos *et al.*, 2017; Giatrakos *et al.*, 2020].

The Event Calculus for Run-Time Reasoning (RTEC) [Artikis *et al.*, 2015] is a composite event recognition (CER) system that has been tested and proven efficient in numerous applications, such as urban traffic management, public space surveillance and maritime situational awareness [Patroumpas *et al.*, 2017]. RTEC is a dialect of the Event Calculus, which is a logic programming formalism for representing and reasoning about events and their effects [Kowalski and Sergot, 1986]. Moreover, RTEC includes various optimisation techniques for computing the intervals of CEs in a data stream.

In real-life streaming tasks, the usual case is for the input events to arrive with variable delays to the CER system. In the maritime domain, for example, delays occur in the input events when the stations (terrestrial and satellite) that collect the position signals of vessels have to deal with a great amount of messages. Furthermore, revisions or retractions may be applied to input events; e.g. the start or end time of an event may be wrong and subsequently corrected by the event source.

Delays and retractions in the input stream are being handled by RTEC by means of windowing. RTEC employs overlapping windows, in order to 'wait' for delayed events and retractions. A drawback is that CE intervals within a window are computed from scratch, without considering the CE intervals of the previous overlapping windows. This way the intervals of a CE will be re-calculated even if delays and retractions do not have an effect on them. In [Tsilionis *et al.*, 2022], we present $RTEC_{inc}$, an incremental version of RTEC, that overcomes this type of inefficiency. In particular, our contributions are the following:

- We present an incremental algorithm for all types of CEs of the language of RTEC.
- We provide a detailed theoretical evaluation of $RTEC_{inc}$ 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 from different application domains. The results are compatible with our complexity analysis and show that $RTEC_{inc}$ is preferable over RTEC in many practical cases. The code of $RTEC_{inc}$ is publicly available[1]. Moreover, some of the employed datasets are available, allowing the reproducibility of our results.

The structure of this paper is as follows: Section 2 summarises the functionality of RTEC. Section 3 outlines the details of the incremental procedure. Finally, Section 4 summarises our empirical analysis.

[*]Efthimis Tsilionis, Alexander Artikis, and Georgios Paliouras. Incremental event calculus for run-time reasoning. *J. Artif. Intell. Res.*, 73:967–1023, 2022.

[1]https://github.com/eftsilio/Incremental_RTEC

2 Background: Run-Time Event Calculus

2.1 Language & Semantics

The time model used by RTEC is linear and includes integer time-points [Artikis *et al.*, 2015]. If F is a *fluent* — a property that can 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. Event Calculus *events* express instantaneous SDEs and CEs, while fluent-value pairs express durative SDEs and CEs. In CER, the majority of CEs are durative and, therefore, the task is to compute the maximal intervals for which a fluent-value pair $F{=}V$ representing a CE has a particular value continuously. Fluents in RTEC are of two kinds: simple and statically determined. In this paper, we restrict attention to simple fluents.

Simple fluents are defined by means of initiatedAt and terminatedAt rules. Below we present an abstract initiatedAt rule. terminatedAt rules have a similar form.

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

Rule (1) is a rule of conjunctions, meaning that all body literals should be satisfied in order for the rule to fire. not denotes negation by failure [Clark, 1977]. Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. The variable T, present at the head and all body literals, expresses that all literals are evaluated at the same time-point. Rule (1) is satisfied at time-point T if event A has occurred at T, there exists an interval of fluent B that includes T, there is no occurrence of event C at T and there is no interval of fluent D that includes T. We use the term *positive* to refer to events and fluents that must occur at or include T, e.g. A and B, and the term *negative* to refer to events and fluents that should not occur at or include T (symbol not), e.g. C and D. 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. 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.

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

CE definitions in RTEC are (locally) stratified logic programs [Przymusinski, 1987].

2.2 Operation

The CER 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 user-specified interval, the *'working memory'* or *window* ω. All SDEs outside the window are discarded and not considered during recognition. This means that at each q_i CER depends only on the SDEs that took place in the interval $(q_i{-}\omega, q_i]$. The size of ω, as well as the temporal distance between two consecutive query times — the 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 step, i.e. $q_i{-}\omega < q_{i-1} < q_i$.

At each query time q_i, RTEC computes from scratch and stores the intervals of fluent-value pairs expressing CEs. Figure 1 illustrates the process of computing the initiation points of the fluent-value pair $F{=}V$ at two consecutive query times with the use of rule (1). In this example, the window ω is longer than the step. At the upper part of Figure 1 the upward arrows represent the initiation points calculated at the previous query time q_{i-1}.

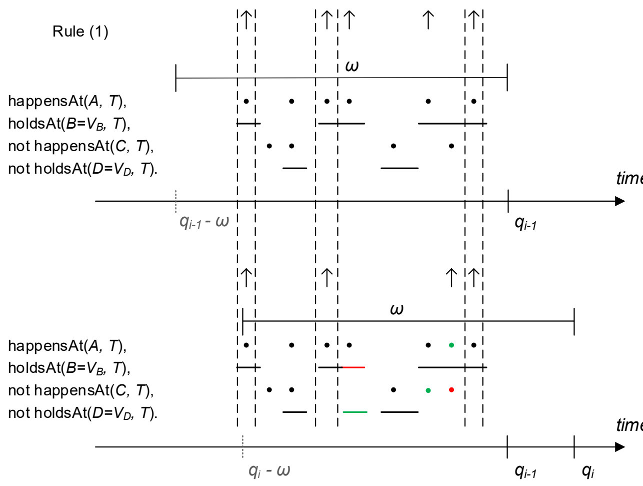


Figure 1: Example of initiation point computation. The upper part of the figure shows the initiation points of fluent-value pair $F{=}V$, as defined by rule (1), calculated at the previous query time q_{i-1}, while the bottom part shows the initiation points calculated at q_i. Dots represent event occurrences,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

At the bottom part of Figure 1, we present the initiation points computed at q_i. Notice the presence of delayed arrivals for events A and C (green dots), of a new interval computed for fluent D (green line), of a retraction of event C (red dot), and of an interval reduction for fluent 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.

However, three out of the four initiation points calculated at q_i are identical among the two query times (see the vertical dashed lines). Although these initiation points are not affected by delays and retractions, they still need to be recomputed. This is unnecessary and indicates the redundant computations of RTEC.

3 Incremental Evaluation of Simple Fluents

We present $RTEC_{inc}$, an extension of RTEC that includes a process for the incremental computation of fluent-value pairs. In this paper, we will summarise the treatment of simple fluents. In what follows, we assume that windows are overlapping, i.e. that $q_i{-}\omega < q_{i-1} < q_i$. Moreover, incremental computation concerns the overlap between consecutive windows, i.e. $(q_i{-}\omega, q_{i-1}]$. Reasoning in $(q_{i-1}, q_i]$ may be performed using RTEC. The incremental evaluation comprises two phases: deletion and addition. During deletion, $RTEC_{inc}$ removes initiation and termination points, calculated at q_{i-1} and no longer holding at q_i. In the addition phase, $RTEC_{inc}$ calculates new initiation and termination points, i.e. points that were not present at q_{i-1}. In the present paper, we will provide only a brief description of the addition phase. The details of both phases, as well as the treatment of statically determined fluents, can be found in the original paper [Tsilionis *et al.*, 2022].

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 two consecutive windows, $(q_i{-}\omega, q_{i-1}]$, or to the non-overlapping part, $(q_{i-1}, q_i]$. We focus on the overlapping part, since it differentiates the method of computation of $RTEC_{inc}$ from that of RTEC.

Consider rule (1) again and assume that at q_{i-1} the rule did not fire, but all body predicates except the first one were satisfied at time-point T. At q_i a delayed arrival of event A at time-point 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.

To calculate the new initiation points, we use the *delta* rules presented in (2), in the given order (termination points are handled similarly). The superscripts correspond to the set in which the time argument T is evaluated. In rule (2)(a), event A is evaluated over the occurrences that arrived at $RTEC_{inc}$ at q_i (set I^+). The time-points in set I^+ are examined against all the intervals of $B{=}V_B$ (set $I^{\mathsf{Q_i}}$) overlapping the interval $(q_i{-}\omega, q_{i-1}]$. If an interval of $B{=}V_B$ includes a time-point in set I^+, then this time-point should not coincide with any occurrence of event C, and should not overlap any of the intervals of $D{=}V_D$. If all of these conditions are satisfied, then the rule gives rise to a new initiation point. Figure 2(a) illustrates the calculation of an initiation point belonging to $(q_i{-}\omega, q_{i-1}]$ by means of rule (2)(a).

$$\begin{array}{llll}
\mathsf{initiatedAt}(F{=}V,\, T) \leftarrow & & \mathsf{initiatedAt}(F{=}V,\, T) \leftarrow & \\
\quad \Big[\mathsf{happensAt}(A,\, T)\Big]^{I^+}, & & \quad \Big[\mathsf{happensAt}(A,\, T)\Big]^{I^{\mathsf{Q_i}}\setminus I^+}, & \\
\quad \Big[\mathsf{holdsAt}(B{=}V_B,\, T)\Big]^{I^{\mathsf{Q_i}}}, & \text{(a)} & \quad \Big[\mathsf{holdsAt}(B{=}V_B,\, T)\Big]^{I^+}, & \text{(b)} \\
\quad \mathsf{not}\ \Big[\mathsf{happensAt}(C,\, T)\Big]^{I^{\mathsf{Q_i}}}, & & \quad \mathsf{not}\ \Big[\mathsf{happensAt}(C,\, T)\Big]^{I^{\mathsf{Q_i}}}, & \\
\quad \mathsf{not}\ \Big[\mathsf{holdsAt}(D{=}V_D,\, T)\Big]^{I^{\mathsf{Q_i}}}. & & \quad \mathsf{not}\ \Big[\mathsf{holdsAt}(D{=}V_D,\, T)\Big]^{I^{\mathsf{Q_i}}}. & \\
\mathsf{initiatedAt}(F{=}V,\, T) \leftarrow & & \mathsf{initiatedAt}(F{=}V,\, T) \leftarrow & \\
\quad \Big[\mathsf{happensAt}(C,\, T)\Big]^{I^-}, & & \quad \Big[\mathsf{happensAt}(A,\, T)\Big]^{I^{\mathsf{Q_i}}\setminus I^+}, & \\
\quad \Big[\mathsf{happensAt}(A,\, T)\Big]^{I^{\mathsf{Q_i}}\setminus I^+}, & \text{(c)} & \quad \Big[\mathsf{holdsAt}(D{=}V_D,\, T)\Big]^{I^-}, & \text{(d)} \\
\quad \Big[\mathsf{holdsAt}(B{=}V_B,\, T)\Big]^{I^{\mathsf{Q_i}}\setminus I^+}, & & \quad \Big[\mathsf{holdsAt}(B{=}V_B,\, T)\Big]^{I^{\mathsf{Q_i}}\setminus I^+}, & \\
\quad \mathsf{not}\ \Big[\mathsf{holdsAt}(D{=}V_D,\, T)\Big]^{I^{\mathsf{Q_i}}}. & & \quad \mathsf{not}\ \Big[\mathsf{happensAt}(C,\, T)\Big]^{I^{\mathsf{Q_i}}\cup I^-}. &
\end{array} \qquad (2)$$

Rule (2)(b) is similar to (2)(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 I^+). However, event A is matched against the occurrences at q_i, excluding the occurrences that were inserted to the system at q_i (set $I^{\mathsf{Q_i}} \setminus I^+$). This means that we examine only time-points falling in $(q_i{-}\omega, q_{i-1}]$ and present in the system from the previous query time q_{i-1}, excluding the ones, if any, that were retracted at q_i. This is important in order to avoid repeating evaluations. Figure 2(b) shows the calculation of an initiation point belonging to $(q_i{-}\omega, q_{i-1}]$ by using rule (2)(b).

Rule (2)(c) examines if a retracted occurrence of event C (set I^-) 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 some occurrences of event C are removed, an initiation point of $F{=}V$ could have been calculated. Notice that the conditions of the rule have been re-ordered for performance reasons. Figure 2(c) shows this process by using rule (2)(c).

Rule (2)(d) examines the deleted intervals of $D{=}V_D$. We employ the same optimisations as in the previous two rules, but we also introduce a new one concerning negative literals. In rule (2)(c) we examined event C over the time-points in I^-. If any of these points led to new initiation points, then these derivations should not be repeated in rule (2)(d). In order to achieve this, we evaluate event C negatively over all of its occurrences at q_i, including the deleted ones (set $I^{\mathsf{Q_i}} \cup I^-$). In other words, we take into account the occurrences in I^-, which are not present at q_i, in order to avoid repeating derivations. Figure 2(d) shows the calculation of an initiation point belonging to $(q_i{-}\omega, q_{i-1}]$ by means of rule (2)(d).

In each of the four *delta* rules, a body literal is evaluated over the set I^+ or I^-. In practice these sets are small, compared to the set of all event occurrences and fluent intervals, i.e. $I^{\mathsf{Q_i}}$. By using these small sets, the evaluation is faster and the performance is improved compared to the re-computation from scratch performed by RTEC. The optimised evaluation sets of the *delta* rules also ensure that a new initiation point

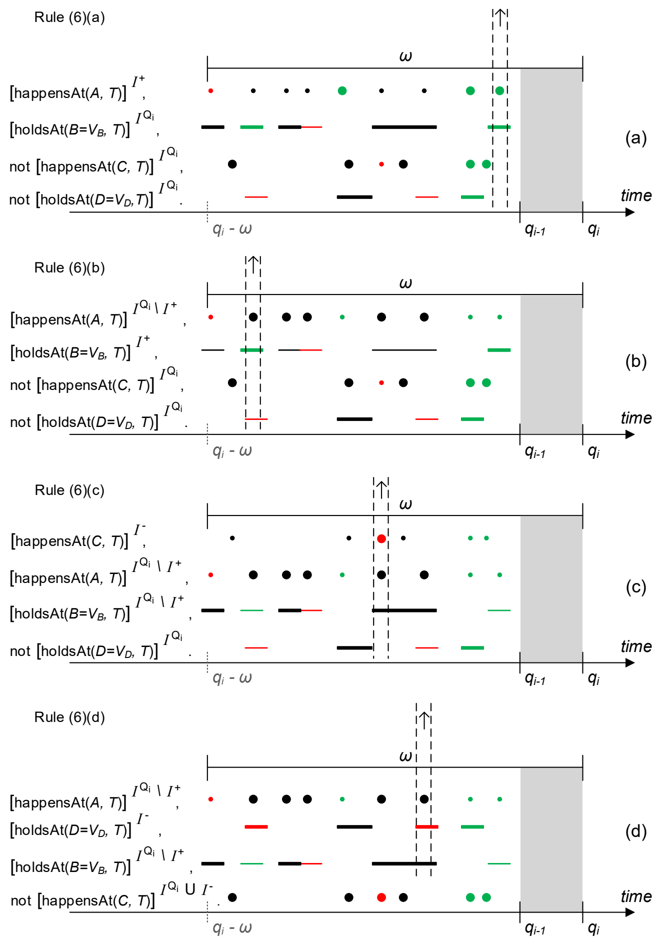

Rule (6)(a)
ω
[happensAt(A, T)] I^+,
[holdsAt(B=V_B, T)] I^{Q_i},
not [happensAt(C, T)] I^{Q_i},
not [holdsAt(D=V_D,T)] I^{Q_i}.
(a)
time
q_i - ω
q_{i-1}
q_i
Rule (6)(b)
[happensAt(A, T)] $I^{Q_i} \setminus I^+$,
[holdsAt(B=V_B, T)] I^+,
not [happensAt(C, T)] I^{Q_i},
not [holdsAt(D=V_D, T)] I^{Q_i}.
(b)
Rule (6)(c)
[happensAt(C, T)] I^-,
[happensAt(A, T)] $I^{Q_i} \setminus I^+$,
[holdsAt(B=V_B, T)] $I^{Q_i} \setminus I^+$,
not [holdsAt(D=V_D, T)] I^{Q_i}.
(c)
Rule (6)(d)
[happensAt(A, T)] $I^{Q_i} \setminus I^+$,
[holdsAt(D=V_D, T)] I^-,
[holdsAt(B=V_B, T)] $I^{Q_i} \setminus I^+$,
not [happensAt(C, T)] $I^{Q_i} \cup I^-$.
(d)

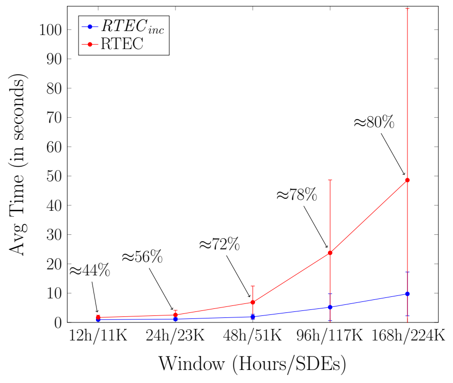
Figure 2: Illustration of the addition phase. Dots represent event occurrences, unlabeled horizontal lines represent fluents intervals and arrows facing upwards represent initiation points. The black color signifies event occurrences and fluent intervals present both at q_{i-1} and q_i, the green color signifies events arriving at, and fluent intervals computed at q_i, while the red color signifies event occurrences and fluent intervals retracted at q_i. Enlarged dots and lines denote participation in the addition phase. The vertical dashed lines now indicate the time-points and intervals that give rise to a new initiation point. Each of the four illustrations corresponds to a *delta* rule of rule-set (2). The non-overlapping part of the two query times, $(q_{i-1}, q_i]$, is greyed out in all illustrations.

can only be produced by one of the four rules.

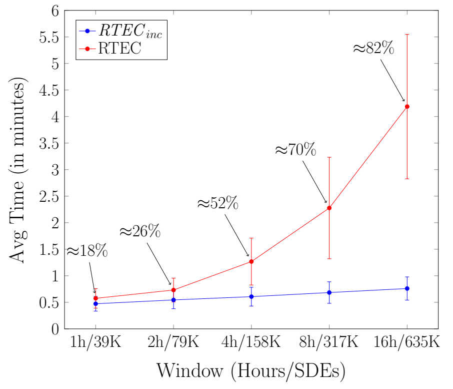
4 Empirical Analysis

We summarise the experimental comparison of RTEC and $RTEC_{inc}$ in the field of maritime monitoring. Maritime situational awareness concerns the recognition of compostite maritime events, such as ship-to-ship transfer, fishing and dangerous sailing, and is typically achieved by monitoring the messages vessels emit while sailing at sea. Terrestrial and satellite stations collect and forward the messages emitted by the vessels to the CER system. The stations have to deal with a large number of messages that lead to significant delays. The CER system must handle in an efficient way the delayed arrival of events in order to minimise latency. We present results from two datasets, a publicly available dataset concerning vessels sailing in the Atlantic Ocean around the port of Brest, France, and a dataset concerning vessels sailing in the European seas. The delays in the first dataset cannot be recovered and an approach of artificially injecting delays was adopted. We show the results of the experiment where 40% of the total events are delayed. In the second dataset we are able to retain the natural delays. The full experimental evaluation can be found in the original paper [Tsilionis *et al.*, 2022].

Figure 3 presents the average recognition times for the synthetic (a) and natural delays dataset (b). In Figure 3(a) the sliding window varies from 12 to 168 hours, while the slide step is constant and equal to 12 hours. In Figure 3(b), the window varies from 1 to 8 hours and the step is equal to 1 hour. In both figures, we state for each window the average number of SDEs that it contains. $RTEC_{inc}$ achieves a performance gain for all window sizes in both datasets. The numbers next to the arrows in Figures 3(a) and (b) denote the approximate improvement $RTEC_{inc}$ brings compared to RTEC. The performance gain becomes more profound as the sliding window size increases. Furthermore, $RTEC_{inc}$ has more predictable performance, as indicated by the standard deviations. The empirical analysis is consistent with the complexity analysis, which is presented in detail in the original paper [Tsilionis *et al.*, 2022].



(a) Synthetic delays (slide step of 12 hours).



(b) Natural delays (slide step of 1 hour).

Figure 3: Average recognition time.

References

[Alevizos *et al.*, 2017] Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. Probabilistic complex event recognition: A survey. *ACM Comput. Surv.*, 50(5):71:1–71:31, 2017.

[Artikis *et al.*, 2015] Alexander Artikis, Marek J. Sergot, and Georgios Paliouras. An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng.*, 27(4):895–908, 2015.

[Clark, 1977] Keith L. Clark. Negation as failure. In *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, pages 293–322, 1977.

[Cugola and Margara, 2012] Gianpaolo Cugola and Alessandro Margara. Complex event processing with t-rex. *Journal of Systems and Software*, 85(8):1709 – 1728, 2012.

[Giatrakos *et al.*, 2020] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020.

[Kowalski and Sergot, 1986] Robert A. Kowalski and Marek J. Sergot. A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95, 1986.

[Patroumpas *et al.*, 2017] Kostas Patroumpas, Elias Alevizos, Alexander Artikis, Marios Vodas, Nikos Pelekis, and Yannis Theodoridis. Online event recognition from moving vessel trajectories. *GeoInformatica*, 21(2):389–427, 2017.

[Przymusinski, 1987] T. Przymusinski. On the declarate semantics of stratified deductive databases and logic programs. In *Foundations of Deductive Databases and Logic Programming*. Morgan, 1987.

[Tsilionis *et al.*, 2022] Efthimis Tsilionis, Alexander Artikis, and Georgios Paliouras. Incremental event calculus for run-time reasoning. *J. Artif. Intell. Res.*, 73:967–1023, 2022.