Complex Event Recognition with Allen Relations

Periklis Mantenoglou[1,2] , **Dimitrios Kelesis**[3,2] , **Alexander Artikis**[4,2]
[1]National and Kapodistrian University of Athens, GR
[2]NCSR "Demokritos", GR
[3]National Technical University of Athens, GR
[4]University of Piraeus, GR
periklismant@di.uoa.gr, dkelesis@mail.ntua.gr, a.artikis@unipi.gr

Abstract

Contemporary applications require the processing of large, high-velocity streams of symbolic events derived from sensor data. A complex event recognition (CER) system processes these symbolic events online and reports the satisfaction of complex event patterns with minimal latency. We extend an Event Calculus dialect optimised for online CER with Allen's interval algebra, in order to provide more accurate event patterns. We demonstrate the effectiveness of our system on real data streams from maritime situational awareness.

1 Introduction

Complex event recognition (CER) systems process high-velocity data streams in order to extract and report instances of (spatio-)temporal pattern satisfaction with minimal latency (Cugola and Margara 2012). CER has been applied to various contemporary applications. In maritime situational awareness, e.g., a CER system consumes streams of vessel position signals, in order to detect instances of dangerous, suspicious and illegal vessel activities in real time, thus supporting safe shipping (Pitsikalis et al. 2019).

The target activities of a CER system, such as illegal fishing, are typically durative, and thus should be expressed using temporal intervals. Such a treatment allows the detection of ongoing activities, while avoiding the unintended semantics of using single time-points (Giatrakos et al. 2020). Moreover, the use of Allen's Interval Algebra has proven quintessential for CER (Awad et al. 2022; Körber et al. 2019; Song et al. 2013; Anicic et al. 2012; Brendel et al. 2011). Allen's algebra specifies thirteen jointly exhaustive and pairwise disjoint relations among intervals (Allen 1984). Consider, e.g., the detection of a 'suspicious rendez-vous' of two vessels in the maritime domain, where one of the vessels turns off signal transmissions, while being close to the other vessel. This phenomenon can be expressed with the 'during' relation of Allen's algebra, while it cannot be captured by common interval operators, such as union and intersection.

We extend the Event Calculus for Run-Time reasoning (RTEC) (Mantenoglou et al. 2022), a formal, logic-based CER system, in order to support the relations of Allen's interval algebra in temporal patterns. RTEC already includes optimisation techniques, like windowing, allowing for highly efficient reasoning in CER applications (Mantenoglou et al. 2022; Tsilionis et al. 2022).

The contributions of this paper may be summarised as follows. First, we present RTEC_A, an open source, formal computational framework for CER with Allen relations. We present the syntax, semantics and reasoning algorithms of RTEC_A. Second, we prove the correctness of RTEC_A, which stems from the use of a novel interval caching technique. Third, we show that RTEC_A has linear-time complexity, bound by a small part of the stream of constant size. Fourth, we present an extensive, reproducible empirical comparison of our approach with two state-of-the-art systems supporting Allen relations on real maritime data. Our comparison demonstrates that RTEC_A is at least one order of magnitude more efficient than the state-of-the-art.

2 Background

2.1 Event Calculus for Run-Time Reasoning

The Event Calculus for Run-Time reasoning (RTEC) (Mantenoglou et al. 2022; Artikis et al. 2015) is a logic programming implementation of the Event Calculus (Kowalski and Sergot 1986), designed for reasoning over data streams. The time model of RTEC is linear and includes integer time-points. Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. The language of RTEC includes events and fluents, i.e., properties that may have different values at different points in time. The term $F{=}V$ denotes that fluent F has value V. Boolean fluents are a special case where the possible values are true and false. Events are instantaneous and may change the values of fluents. The task of RTEC is to compute the maximal intervals during which a fluent-value pair (FVP) holds continuously. The main predicates of RTEC are the following. $\mathsf{happensAt}(E, T)$ denotes that event E occurs at time-point T, while $\mathsf{initiatedAt}(F = V, T)$ (resp. $\mathsf{terminatedAt}(F = V, T)$) specifies that fluent F starts (stops) having value V at T. For a FVP $F{=}V$, $\mathsf{holdsFor}(F = V, I)$ states that fluent F has value V continuously during the maximal intervals in list I, and $\mathsf{holdsAt}(F = V, T)$ expresses the truth value of $F{=}V$ at a given time-point T.

A formalisation in RTEC contains a set of application-specific rules, expressing the relations between the events

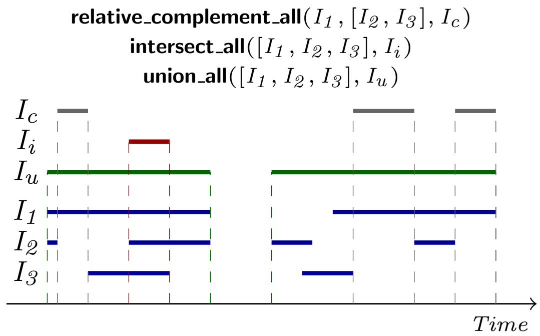


Figure 1: Interval manipulation constructs of RTEC. I_1, I_2 and I_3 (resp. I_c, I_i and I_u) are input (output) lists of maximal intervals.

and the FVPs of a domain, called *event description*.

Definition 1 (Event description)**.** An event description is a set of:

- Ground facts in the form of $\mathsf{happensAt}(E, T)$, expressing a stream of event instances.
- Rules with head $\mathsf{initiatedAt}(F = V, T)$ or $\mathsf{terminatedAt}(F = V, T)$, expressing the effects of events on FVPs.
- Rules with head $\mathsf{holdsFor}(F = V, I)$, defining $F{=}V$ in terms of other FVPs. ■

RTEC features *simple* and *statically determined* fluents. Simple fluents are defined by means of initiatedAt and terminatedAt rules, and are subject to the commonsense law of inertia, i.e., a FVP $F{=}V$ holds at a time-point T, if $F{=}V$ has been 'initiated' by an event at a time-point earlier than T, and not 'terminated' by another event in the meantime.

Example 1 (Within area)**.** In maritime monitoring, various areas, e.g., fisheries restricted areas, disallow certain activities. It is thus useful to compute the intervals during which a vessel is in such an area. See the formalisation below:

$$\begin{aligned}&\mathsf{initiatedAt}(withinArea(Vl, AreaType) = \mathsf{true},\ T) \leftarrow\\ &\quad \mathsf{happensAt}(entersArea(Vl, AreaID), T),\\ &\quad areaType(AreaID, AreaType).\end{aligned} \tag{1}$$

$$\begin{aligned}&\mathsf{terminatedAt}(withinArea(Vl, AreaType) = \mathsf{true},\ T) \leftarrow\\ &\quad \mathsf{happensAt}(leavesArea(Vl, AreaID), T),\\ &\quad areaType(AreaID, AreaType).\end{aligned} \tag{2}$$

$$\begin{aligned}&\mathsf{terminatedAt}(withinArea(Vl, AreaType) = \mathsf{true},\ T) \leftarrow\\ &\quad \mathsf{happensAt}(gapStart(Vl), T).\end{aligned} \tag{3}$$

$withinArea(Vl, AreaType)$ is a Boolean simple fluent denoting that a vessel Vl is in some area of $AreaType$, while $entersArea(Vl, AreaID)$, $leavesArea(Vl, AreaID)$ and $gapStart(Vl)$ are input events, derived by the online processing of vessel position signals, and their spatial relations with areas of interest (Santipantakis et al. 2018). $areaType(AreaID, AreaType)$ is an atemporal predicate storing background knowledge concerning the types of areas in a dataset. Rules (1) and (2) state that $withinArea(Vl, AreaType)$ is initiated (resp. terminated) as soon as vessel Vl enters (leaves) an area $AreaID$, whose type is $AreaType$. According to rule (3), $withinArea(Vl, AreaType)$ is terminated when there is a communication gap, i.e., when Vl stops transmitting its position. In this case, we are uncertain of the vessel's whereabouts. Using rules (1)-(3), RTEC computes, with the use of application-independent rules, $\mathsf{holdsFor}(withinArea(Vl, AreaType) = \mathsf{true}, I)$, i.e., the list of maximal intervals I during which Vl is in $AreaType$. □

The syntax of simple fluent definitions is provided in the supplementary document. In addition to the domain-independent definition of holdsFor, which is used for computing the maximal intervals of simple fluents, an event description may include domain-specific holdsFor rules, used to define the values of a fluent F in terms of the values of other fluents. Such a fluent F is called 'statically determined', and the maximal intervals of $F = V$ are derived by applying on the intervals of other FVPs the following interval manipulation constructs: union_all, intersect_all and relative_complement_all . $\mathsf{union\_all}(L, I)$ (resp. $\mathsf{intersect\_all}(L, I)$) computes the list of maximal intervals I as the union (intersection) of all lists of maximal intervals of list L. $\mathsf{relative\_complement\_all}(I', L, I)$ computes the list of maximal intervals I by removing from the maximal intervals of list I' all time-points included in some list of maximal intervals of list L. Figure 1 presents an illustration.

Definition 2 (Syntax of statically determined fluent definitions)**.** The rules defining statically determined fluents have the following syntax:

$$\begin{aligned}&\mathsf{holdsFor}(F = V, I_{n+m}) \leftarrow\\ &\quad \mathsf{holdsFor}(F_1 = V_1, I_1)[[, \mathsf{holdsFor}(F_2 = V_2, I_2),\ \ldots\\ &\quad \mathsf{holdsFor}(F_n = V_n, I_n),\ \mathsf{intervalConstruct}(L_1, I_{n+1}),\\ &\quad \ldots,\ \mathsf{intervalConstruct}(L_m, I_{n+m})]].\end{aligned} \tag{4}$$

The first body literal of a holdsFor rule defining $F = V$ is a holdsFor predicate expressing the maximal intervals of a FVP other than $F = V$. This is followed by a possibly empty list, denoted by '[[ ]]', of holdsFor predicates for other FVPs, and interval manipulation constructs, expressed by intervalConstruct in formulation (4). $\mathsf{intervalConstruct}(L_j, I_{n+j})$ may be $\mathsf{union\_all}(L_j, I_{n+j})$, $\mathsf{intersect\_all}(L_j, I_{n+j})$ or $\mathsf{relative\_complement\_all}(I_k, L_j, I_{n+j})$. I_k, where $k < n+j$, is a list of maximal intervals appearing earlier in the body of the rule, and list L_j contains a subset of such lists. The output list I_{n+m} contains the maximal intervals during which $F{=}V$ holds continuously. ■

Example 2 (Anchored and moored vessels)**.** Consider the following example from maritime situational awareness:

$$\begin{aligned}&\mathsf{holdsFor}(anchoredOrMoored(Vl) = \mathsf{true}, I) \leftarrow\\ &\quad \mathsf{holdsFor}(stopped(Vl) = farFromPorts, I_{sf}),\\ &\quad \mathsf{holdsFor}(withinArea(Vl, anchorage) = \mathsf{true}, I_a),\\ &\quad \mathsf{intersect\_all}([I_{sf}, I_a], I_{sfa}),\\ &\quad \mathsf{holdsFor}(stopped(Vl) = nearPorts, I_{sn}),\\ &\quad \mathsf{union\_all}([I_{sfa}, I_{sn}], I).\end{aligned} \tag{5}$$

$anchoredOrMoored(Vl)$ is a Boolean statically determined fluent defined in terms of the FVPs $stopped(Vl) = farFromPorts$, $stopped(Vl) = nearPorts$ and $withinArea(Vl, anchorage) = \mathsf{true}$. $stopped(Vl)$ is a multi-valued fluent expressing the periods during which

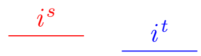
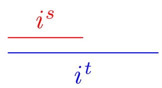
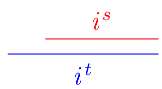
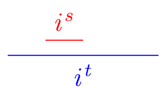
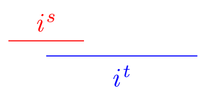
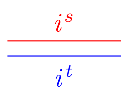
Relation	Definition	Illustration
before(i^s, i^t)	$f(i^s) < s(i^t)$	
meets(i^s, i^t)	$f(i^s) = s(i^t)$	
starts(i^s, i^t)	$s(i^s) = s(i^t)$, $f(i^s) < f(i^t)$	
finishes(i^s, i^t)	$s(i^s) > s(i^t)$, $f(i^s) = f(i^t)$	
during(i^s, i^t)	$s(i^s) > s(i^t)$, $f(i^s) < f(i^t)$	
overlaps(i^s, i^t)	$s(i^s) < s(i^t)$, $f(i^s) > s(i^t)$, $f(i^s) < f(i^t)$	
equal(i^s, i^t)	$s(i^s) = s(i^t)$, $f(i^s) = f(i^t)$	

Table 1: Seven relations of Allen's interval algebra.

vessel Vl is idle near some port or far from all ports. The specification of this fluent is available with the complete event description of maritime situational awareness[1]. Rule (5) derives the intervals during which vessel Vl is both stopped far from all ports and within an anchorage area, by applying the intersect_all operation on the lists of maximal intervals I_{sf} and I_a. The output of this operation is list I_{sfa}. Subsequently, list I is derived by applying union_all on lists I_{sfa} and I_{sn}. In this way, list I contains the maximal intervals during which vessel Vl has stopped near some port or within an anchorage area. □

For a wide range of fluents, the use of union_all, intersect_all and relative_complement_all allows for more concise event descriptions, as opposed to the traditional style of Event Calculus representation, i.e., identifying the various conditions under which a fluent is initiated and terminated, so that maximal intervals can then be computed using the domain-independent holdsFor. Moreover, according to the complexity analysis of (Artikis et al. 2015), the interval manipulation constructs of RTEC can also lead to much more efficient computation. In this paper, we extend the expressivity of the language of RTEC by allowing for Allen's relations within statically determined fluent definitions.

2.2 Allen's Interval Algebra

Allen's interval algebra specifies thirteen jointly exhaustive and pairwise disjoint relations among intervals (Allen 1984). Table 1 presents relations before, meets, starts, finishes, during, overlaps and equal. The remaining six relations are the 'inverse' relations; equal does not have an inverse relation. The second column of Table 1 shows the conditions that must be satisfied in order to compute an Allen relation. i^s and i^t express intervals, while $s(i)$ and $f(i)$ denote the start and end endpoint of interval i, respectively.

[1]https://github.com/aartikis/RTEC

outMode	Output list I
source	$I = \mathcal{S}_{\text{rel}}$
target	$I = \mathcal{T}_{\text{rel}}$
union	union_all$([\mathcal{S}_{\text{rel}}, \mathcal{T}_{\text{rel}}], I)$
intersect	intersect_all$([\mathcal{S}_{\text{rel}}, \mathcal{T}_{\text{rel}}], I)$
complement	relative_complement_all$(\mathcal{S}_{\text{rel}}, [\mathcal{T}_{\text{rel}}], I)$
complement_inv	relative_complement_all$(\mathcal{T}_{\text{rel}}, [\mathcal{S}_{\text{rel}}], I)$

Table 2: Output modes of the allen construct.

Allen's relations have proven necessary for CER (Awad et al. 2022; Körber et al. 2019). However, the interval manipulation constructs of RTEC cannot express the relations of Allen's algebra. Consider, e.g., the computation of the interval pairs (i^s, i^t), where $i^s \in \mathcal{S}$ and $i^t \in \mathcal{T}$, satisfying before. intersect_all$([\mathcal{S}, \mathcal{T}], [\,])$ states that for every interval pair (i^s, i^t), such that $i^s \in \mathcal{S}$, $i^t \in \mathcal{T}$, it holds that $i^s \cap i^t = \emptyset$. Therefore, i^s is before i^t, or vice versa. It is impossible, however, to distinguish between the two cases.

3 Allen Relations in Event Descriptions

We present RTEC$_{\text{A}}$, an extension of RTEC supporting CER specifications requiring Allen relations.

3.1 Representation and Semantics

Representation. We cannot express Allen relations in RTEC without extending its expressive power. Simple fluent definitions evaluate fluent initiation and termination conditions on a particular time instant, and thus do not support interval endpoint comparisons. Moreover, as already mentioned, the interval manipulation constructs in statically determined fluent definitions cannot express Allen relations. To address this issue, we extend the statically determined fluent definitions.

Definition 3 (Syntax of statically determined fluent definitions in RTEC$_{\text{A}}$)**.** A holdsFor$(F = V, I)$ rule defining a statically determined fluent F may additionally contain body predicates in the form of allen(rel, $\mathcal{S}$, $\mathcal{T}$, outMode, I), where rel denotes an Allen relation, $\mathcal{S}$ and $\mathcal{T}$ are input lists of maximal intervals, outMode expresses how we should treat the interval pairs (i^s, i^t) satisfying rel, where $i^s \in \mathcal{S}$ and $i^t \in \mathcal{T}$, and I is the output list of maximal intervals. ■

According to allen(rel, $\mathcal{S}$, $\mathcal{T}$, outMode, I), I contains the maximal intervals produced by applying outMode to the interval pairs of the 'source list' $\mathcal{S}$ and the 'target list' $\mathcal{T}$ satisfying rel, i.e., one of the Allen relations presented in Table 1. The inverse relations may be computed by reversing the order of the input lists. outMode is applied to the intervals of lists $\mathcal{S}_{\text{rel}} = \{i^s \mid i^s \in \mathcal{S} \land \exists i^t \in \mathcal{T} : \text{rel}(i^s, i^t)\}$ and $\mathcal{T}_{\text{rel}} = \{i^t \mid i^t \in \mathcal{T} \land \exists i^s \in \mathcal{S} : \text{rel}(i^s, i^t)\}$, i.e., the intervals of the source and the target lists appearing in at least one pair of intervals satisfying rel. The possible values of outMode and

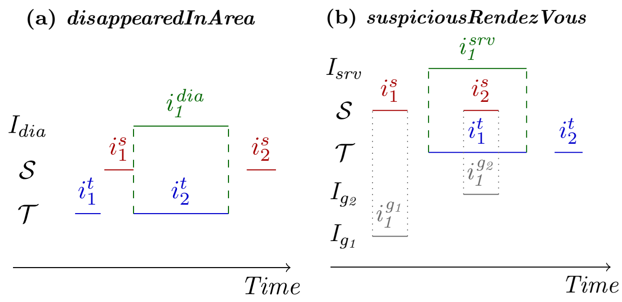


Figure 2: Maximal interval computation with the allen construct.

their meaning are presented in Table 2. Below, we illustrate the use of the allen predicate.

Example 3 (Allen relations for maritime situational awareness)**.** Vessels often attempt to conceal illegal activities in certain areas, such as fishing in fisheries restricted areas, by stopping transmitting their position. See the rule below:

$$
\begin{aligned}
&\mathsf{holdsFor}(\mathit{disappearedInArea}(\mathit{Vl}, \mathit{AreaType}) = \mathsf{true}, I_{dia}) \leftarrow \\
&\quad \mathsf{holdsFor}(\mathit{withinArea}(\mathit{Vl}, \mathit{AreaType}) = \mathsf{true}, \mathcal{S}), \\
&\quad \mathsf{holdsFor}(\mathit{gap}(\mathit{Vl}) = \mathit{farFromPorts}, \mathcal{T}), \\
&\quad \mathsf{allen}(\mathsf{meets}, \mathcal{S}, \mathcal{T}, \mathsf{target}, I_{dia}).
\end{aligned} \tag{6}
$$

$\mathit{disappearedInArea}(\mathit{Vl}, \mathit{AreaType})$ is a statically determined Boolean fluent defined in terms of $\mathit{withinArea}(\mathit{Vl}, \mathit{AreaType})$ (see Example 1), and $\mathit{gap}(\mathit{Vl})$, i.e., a multi-valued fluent expressing the intervals during which vessel Vl stopped transmitting its position. The specification of *gap* is available with the complete event description of maritime situational awareness[1]. The last condition of rule (6) expresses the meets Allen relation. $\mathsf{allen}(\mathsf{meets}, \mathcal{S}, \mathcal{T}, \mathsf{target}, I_{dia})$ states that from the interval pairs (i^s, i^t) satisfying meets, where $i^s \in \mathcal{S}$ and $i^t \in \mathcal{T}$, we will keep in the output list I_{dia} the target intervals i^t (see the second line of Table 2). According to rule (6), therefore, a vessel Vl is said to disappear in an area of $\mathit{AreaType}$ during an interval i^{dia}, if i^{dia} is an interval during which $\mathit{gap}(\mathit{Vl}) = \mathit{farFromPorts}$, i.e., Vl stopped transmitting its position while being in the open sea, and i^{dia} is met by an interval i^s during which Vl was within an area of $\mathit{AreaType}$. Figure 2(a) provides a graphical illustration. In this illustration, the only interval pair satisfying meets is (i^s_1, i^t_2), and thus $I_{dia} = [i^t_2]$. If we wanted to include i^s_1 in the output I_{dia}, then we would have replaced target by union in the last condition of rule (6) (see the third line of Table 2). This way, i^s_1 would be amalgamated with i^t_2 producing a single interval, i.e., $I_{dia} = [i^s_1 \cup i^t_2]$. In any case, the interval manipulation constructs of RTEC cannot express *disappearedInArea*. For instance, $\mathsf{relative\_complement\_all}(\mathcal{T}, [\mathcal{S}], I)$ would discard the common time-point of i^s_1 and i^t_2, and would include i^t_1, which does not satisfy meets. Similarly, $\mathsf{union\_all}([\mathcal{S}, \mathcal{T}], I_{dia})$ would include all intervals of $\mathcal{S}$ and $\mathcal{T}$, which is incorrect.

Proximate vessels may stop transmitting their position to conduct illegal activities, such as an illegal cargo transfer. Consider the formalisation below:

$$
\begin{aligned}
&\mathsf{holdsFor}(\mathit{suspiciousRendezVous}(\mathit{Vl_1}, \mathit{Vl_2}) = \mathsf{true}, I_{srv}) \leftarrow \\
&\quad \mathsf{holdsFor}(\mathit{gap}(\mathit{Vl_1}) = \mathit{farFromPorts}, I_{g_1}), \\
&\quad \mathsf{holdsFor}(\mathit{gap}(\mathit{Vl_2}) = \mathit{farFromPorts}, I_{g_2}), \\
&\quad \mathsf{holdsFor}(\mathit{proximity}(\mathit{Vl_1}, \mathit{Vl_2}) = \mathsf{true}, \mathcal{T}), \\
&\quad \mathsf{union\_all}([I_{g_1}, I_{g_2}], \mathcal{S}), \mathsf{allen}(\mathsf{during}, \mathcal{S}, \mathcal{T}, \mathsf{target}, I_{srv}).
\end{aligned} \tag{7}
$$

$\mathit{suspiciousRendezVous}(\mathit{Vl_1}, \mathit{Vl_2})$ is a statically determined fluent, and $\mathit{proximity}(\mathit{Vl_1}, \mathit{Vl_2})$ is a Boolean fluent denoting whether two vessels, $\mathit{Vl_1}$ and $\mathit{Vl_2}$, are close to each other. $\mathcal{T}$, i.e., the list of maximal intervals during which two vessels are close to each other, is derived by an online spatial processing technique on vessel positional signals, which is robust to interim signal gaps (Santipantakis et al. 2018). union_all in rule (7) derives $\mathcal{S}$, i.e., the list of maximal intervals during which $\mathit{gap}(\mathit{Vl_1}) = \mathit{farFromPorts}$ or $\mathit{gap}(\mathit{Vl_2}) = \mathit{farFromPorts}$. Then, $\mathsf{allen}(\mathsf{during}, \mathcal{S}, \mathcal{T}, \mathsf{target}, I_{srv})$ identifies the maximal intervals of $\mathcal{T}$ that contain an interval of $\mathcal{S}$ and stores them in list I_{srv}. Therefore, rule (7) specifies that vessels $\mathit{Vl_1}$ and $\mathit{Vl_2}$ may be conducting an illegal activity during an interval i^{srv}, if i^{srv} is an interval during which $\mathit{Vl_1}$ and $\mathit{Vl_2}$ are close to each other, and i^{srv} contains an interval i^s during which at least one of the vessels stops transmitting its position. Figure 2(b) displays an illustration of rule (7). union_all constructs list $\mathcal{S}$ as the union of the intervals in lists I_{g_1} and I_{g_2}. Among the intervals of $\mathcal{S}$ and $\mathcal{T}$, during is only satisfied for the interval pair (i^s_2, i^t_1), resulting in $I_{srv} = [i^t_1]$. Note that I_{srv} cannot be derived by the interval manipulation constructs of RTEC. $\mathsf{union\_all}([\mathcal{S}, \mathcal{T}], I)$, e.g., would compute list $I = [i^s_1, i^t_1, i^t_2]$, including intervals i^s_1 and i^t_2 that do not satisfy during. □

As mentioned earlier, Table 2 lists the possible values of outMode and their meaning. Consider the computation of $\mathsf{allen}(\mathsf{meets}, \mathcal{S}, \mathcal{T}, \mathsf{outMode}, I_{dia})$ in the example of Figure 2(a), where the only interval pair satisfying meets is (i^s_1, i^t_2). If outMode was complement or complement_inv, we would apply the relative_complement_all construct on i^s_1 and i^t_2 (complement_inv reverses the order of operants in relative_complement_all) and compute, respectively, list I_{dia} as $[i^s_1 \setminus i^t_2]$ or $[i^t_2 \setminus i^s_1]$. Note that some combinations of rel and outMode are equivalent. For instance, if an interval pair (i^s, i^t) satisfies during, then i^s is a sub-interval of i^t. Therefore, $\mathsf{allen}(\mathsf{during}, \mathcal{S}, \mathcal{T}, \mathsf{union}, I)$ and $\mathsf{allen}(\mathsf{during}, \mathcal{S}, \mathcal{T}, \mathsf{target}, I)$ produce the same output list.

Semantics. An event description in RTEC$_\mathrm{A}$ defines a *dependency graph* expressing the relationships between the FVPs of the event description.

Definition 4 (Dependency Graph)**.** The dependency graph of an event description is a directed graph such that:

1. Each vertex denotes a FVP $F = V$;
2. There exists an edge $(F_j = V_j, F_i = V_i)$ iff:
   - There is an initiatedAt or terminatedAt rule for $F_i = V_i$ having $\mathsf{holdsAt}(F_j = V_j, T)$ as one of its conditions.
   - There is a holdsFor rule for $F_i = V_i$ having $\mathsf{holdsFor}(F_j = V_j, I)$ as one of its conditions. ■

According to Definition 4, the addition of allen constructs

[1]

in statically determined fluents definitions does not introduce additional dependencies among FVPs. Therefore, our extension of RTEC does not affect its semantics.

Proposition 1 (Semantics of RTEC$_\mathsf{A}$)**.** An event description in RTEC$_\mathsf{A}$ is a locally stratified logic program. ▲

A discussion of the semantics of RTEC may be found in (Mantenoglou et al. 2022).

3.2 Reasoning

We extend the process of statically determined fluent evaluation of RTEC with algorithms computing Allen relations. Algorithm 1 presents the steps of the evaluation of $\mathsf{allen}(\mathsf{rel}, \mathcal{S}, \mathcal{T}, \mathsf{outMode}, I)$. This algorithm derives all interval pairs (i^s, i^t), such that $i^s \in \mathcal{S}$ and $i^t \in \mathcal{T}$, satisfying Allen relation rel, and stores them in list *pairs* (line 3). We then compute lists $\mathcal{S}_\mathsf{rel}$ and $\mathcal{T}_\mathsf{rel}$ containing, respectively, the source and the target intervals appearing in list *pairs* at least once (line 4). This way, we may apply $\mathsf{outMode}$ to lists $\mathcal{S}_\mathsf{rel}$ and $\mathcal{T}_\mathsf{rel}$, as specified in Table 2, in order to compute the output list I (line 5). In what follows, we present the algorithm computing the interval pairs of $\mathcal{S}$ and $\mathcal{T}$ satisfying an Allen relation, and the bookkeeping operations which are necessary for correct Allen relation computation in a streaming setting (see lines 1, 2 and 6 of Algorithm 1).

Allen Relation Computation. Algorithm 2 computes the interval pairs of $\mathcal{S}$ and $\mathcal{T}$ satisfying an Allen relation rel and stores them in list *pairs*. I in $\mathsf{holdsFor}(F = V, I)$ is a *sorted* list of maximal intervals (even if the items of the stream are not sorted) (Artikis et al. 2015). Therefore, $\mathcal{S}$ and $\mathcal{T}$ are also sorted lists of maximal intervals (see Definition 3). We evaluate rel by means of interval endpoint comparisons, following the corresponding definition in Table 1. An element of *pairs* is a tuple of the form $(i^s, \mathcal{T}')$, denoting that the source interval i^s satisfies rel with every interval in the list of target intervals $\mathcal{T}' \subseteq \mathcal{T}$. Using this compact representation, we avoid enumerating all computed interval pairs, without information loss. For example, the tuple $(i^s_1, [i^t_1, i^t_2])$ for a relation rel denotes that $\mathsf{rel}(i^s_1, i^t_1)$ and $\mathsf{rel}(i^s_1, i^t_2)$ hold.

Algorithm 2 uses two pointers, p_s and p_t, to traverse $\mathcal{S}$ and $\mathcal{T}$. If rel is before and, indeed, $\mathsf{before}(i^s, i^t)$ holds, we add the tuple $(i^s, [i^t, \dots, \mathcal{T}[\mathit{length}(\mathcal{T})]])$ to *pairs*, where $\mathcal{T}[\mathit{length}(\mathcal{T})]$ denotes the last interval of $\mathcal{T}$ (line 6). If $\mathsf{before}(i^s, i^t)$ does not hold and $f(i^s) \leq f(i^t)$, i.e., the source interval does not end after the target interval, then i^s is before all target intervals after i^t. Consequently, we add the tuple $(i^s, [\mathcal{T}[p_t{+}1], \dots, \mathcal{T}[\mathit{length}(\mathcal{T})]])$ to *pairs* (line 8). If rel is not before and $\mathsf{rel}(i^s, i^t)$ holds, we simply add (i^s, i^t) to *pairs* (line 9).

Afterwards, Algorithm 2 increments pointer p_s and/or pointer p_t. p_s (resp. p_t) may be incremented only if the current source (target) interval i^s (i^t) cannot satisfy rel with any subsequent target (source) interval. Since $\mathcal{S}$ and $\mathcal{T}$ are sorted lists of maximal intervals, we can check this based on the relative positions of i^s and i^t, and the given relation rel, without iterating over any subsequent interval of $\mathcal{S}$ and $\mathcal{T}$. The conditions in which pointers p_s and p_t may be incremented, while guaranteeing the correct computation of interval pairs satisfying rel, are presented in lines 10 and 12.

Algorithm 1 $\mathsf{allen}(\mathsf{rel}, \mathcal{S}, \mathcal{T}, \mathsf{outMode}, I)$

1: $\mathcal{S}^c, \mathcal{T}^c \leftarrow \mathit{retrieveCachedIntervals}(\,)$
2: $\mathcal{S} \leftarrow \mathit{append}(\mathcal{S}^c, \mathcal{S}),\ \ \mathcal{T} \leftarrow \mathit{append}(\mathcal{T}^c, \mathcal{T})$
3: $\mathit{pairs} \leftarrow \mathit{compute\_allen\_relation}(\mathcal{S}, \mathcal{T}, \mathsf{rel})$
4: $\mathcal{S}_\mathsf{rel}, \mathcal{T}_\mathsf{rel} \leftarrow \mathit{getSourceTargetIntervals}(\mathit{pairs})$
5: $I \leftarrow \mathit{applyOutMode}(\mathcal{S}_\mathsf{rel}, \mathcal{T}_\mathsf{rel}, \mathsf{outMode})$
6: $\mathit{windowing}(\mathcal{S}, \mathcal{T}, \mathsf{rel})$

Algorithm 2 $\mathit{compute\_allen\_relation}(\mathcal{S}, \mathcal{T}, \mathsf{rel})$

1: $p_s \leftarrow \mathit{1}$, $p_t \leftarrow \mathit{1}$, $\mathit{pairs} \leftarrow [\,]$
2: **while** $p_s \leq \mathit{length}(\mathcal{S})$ **or** $p_t \leq \mathit{length}(\mathcal{T})$ **do**
3: $\quad i^s \leftarrow \mathcal{S}[p_s]$, $i^t \leftarrow \mathcal{T}[p_t]$
4: $\quad$ **if** $\mathsf{rel} = \mathsf{before}$ **then**
5: $\quad\quad$ **if** $\mathsf{before}(i^s, i^t)$ **then**
6: $\quad\quad\quad \mathit{pairs.add}((i^s, [i^t, \dots, \mathcal{T}[\mathit{length}(\mathcal{T})]]))$
7: $\quad\quad$ **else if** $f(i^s) \leq f(i^t)$ **then**
8: $\quad\quad\quad \mathit{pairs.add}((i^s, [\mathcal{T}[p_t{+}\mathit{1}], \dots, \mathcal{T}[\mathit{length}(\mathcal{T})]]))$
9: $\quad$ **else if** $\mathsf{rel}(i^s, i^t)$ **then** $\mathit{pairs.add}((i^s, [i^t]))$
10: $\quad$ **if** $f(i^s) \leq f(i^t)$ **or** $(s(i^s) \leq f(i^t)$ **and** $\mathsf{rel} \in \{\mathsf{starts}, \mathsf{finishes}, \mathsf{during}, \mathsf{equal}\})$ **then**
11: $\quad\quad p_s \leftarrow p_s + \mathit{1}$
12: $\quad$ **if** $f(i^s) \geq f(i^t)$ **or** $(f(i^s) \geq s(i^t)$ **and** $\mathsf{rel} \in \{\mathsf{before}, \mathsf{meets}, \mathsf{starts}, \mathsf{overlaps}, \mathsf{equal}\})$ **then**
13: $\quad\quad p_t \leftarrow p_t + \mathit{1}$
14: **return** *pairs*

Example 4 (Allen relation computation)**.** In the example of Figure 2(a), $\mathsf{allen}(\mathsf{meets}, \mathcal{S}, \mathcal{T}, \mathsf{target}, I_{dia})$ is used to compute the maximal intervals of *disappearedInArea* in list I_{dia}. In order to derive these intervals, RTEC$_\mathsf{A}$ computes all interval pairs in lists $\mathcal{S}$ and $\mathcal{T}$ satisfying meets (see line 3 of Algorithm 1). This is achieved with Algorithm 2. In this example, the source list is $\mathcal{S} = [i^s_1, i^s_2]$, the target list is $\mathcal{T} = [i^t_1, i^t_2]$. Initially, p_s points to i^s_1 and p_t points to i^t_1. Algorithm 2 verifies that meets does not hold for the interval pair (i^s_1, i^t_1) in line 9. Next, we check whether pointer p_s needs to be incremented. Since $f(i^s_1) > f(i^t_1)$, the condition in line 10 fails, and thus we do not increment p_s. In contrast, the condition in line 12 succeeds because $f(i^s_1) > f(i^t_1)$. Therefore, we increment p_t (line 13). The interval pair of the next iteration is (i^s_1, i^t_2). $\mathsf{meets}(i^s_1, i^t_2)$ is satisfied and thus we compute the pair $(i^s_1, [i^t_2])$. i^s_1 and i^t_2 cannot satisfy meets with any future interval; consequently, we increment both p_s and p_t. There is no target interval after i^t_2. Thus, Algorithm 2 terminates and returns $(i^s_1, [i^t_2])$. □

Windowing. In order to handle streaming data, CER systems often employ windowing techniques. At each 'query time' q_j, RTEC reasons over the items of an input stream that fall within a specified sliding window $w_j = (q_j{-}\omega, q_j]$, where ω is the size of the window. All elements of the stream that took place before or at $q_j{-}\omega$ are discarded/'forgotten'. This ensures that the cost of reasoning depends on the window size ω and not on the complete stream. The size of ω and the temporal distance between two consecutive query times, i.e., the 'step' $q_j{-}q_{j-1}$, may be manually set or opti-

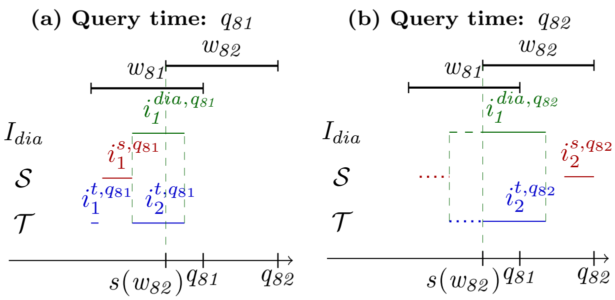


Figure 3: Online maximal interval computation.

mised to meet the requirements of the given application. In the common case that the elements of a stream arrive with delays, e.g., due to network delays, it is preferable to make ω longer than the step. This way, we may reason, at q_j, over the stream elements that took place in $(q_j{-}\omega, q_{j-1}]$, but arrived after q_{j-1}. As an example, Figure 3 shows the intervals of $\mathcal{S}$ and $\mathcal{T}$ of Figure 2(a) as they are available at query times q_{81} and q_{82}. The corresponding windows w_{81} and w_{82} are overlapping in order to accommodate, at query time q_{82}, stream elements that took place in $(q_{82}{-}\omega, q_{81}]$, but arrived after q_{81}.

RTEC$_{\text{A}}$ follows RTEC and reasons over streams using sliding windows. We make the following assumptions. First, the window size and the step remain constant. Thus, we can always derive the endpoints of the next window based on the current query time. Second, the delays in the stream may be tolerated by the window size. In other words, at query time q_j, the intervals taking place before the current window w_j are not revised. In contrast, the intervals that were available or derived at q_{j-1} and take place within w_j may be revised at q_j. RTEC$_{\text{A}}$ guarantees correct reasoning by computing, at q_j, all interval pairs (i^s, i^t) satisfying Allen relation rel, such that at least one of i^s and i^t intersects with window w_j. The proof of correctness is presented in the following section. To compute all such interval pairs, we cache at each query time the intervals that may be required in the future (line 6 of Algorithm 1). This way, we may retrieve at q_j the intervals cached at q_{j-1} (see lines 1–2) that allow us to perform correct Allen relation computation. In the following example, we motivate our caching technique.

Example 5 (Windowing)**.** Figure 3(a) illustrates the computation of allen(meets, $\mathcal{S}$, $\mathcal{T}$, target, I_{dia}) at query time q_{81}, where $\mathcal{S}$ and $\mathcal{T}$ contain only the intervals that fall within window w_{81}. Contrast these intervals with the ones depicted in Figure 2(a). Interval $i_1^{t,q_{81}}$, e.g., is shorter than the interval i^t_1 of Figure 2(a) because a segment of i^t_1 falls outside w_{81}. Moreover, the events leading to the extension of $i_2^{t,q_{81}}$ up to q_{81} have been delayed and are not available at q_{81}, and thus $i_2^{t,q_{81}}$ ends earlier than q_{81}. Based on the intervals in w_{81}, we compute that meets$(i_1^{s,q_{81}}, i_2^{t,q_{81}})$ holds at q_{81} and derive the output interval $i_1^{dia,q_{81}}$, which matches $i_2^{t,q_{81}}$.

The intervals of $\mathcal{S}$ and $\mathcal{T}$ available at the next query time, q_{82}, i.e., $i_2^{s,q_{82}}$ and $i_2^{t,q_{82}}$, are displayed in Figure 3(b). The first segment of $i_2^{t,q_{81}}$, i.e., $[s(i_2^{t,q_{81}}), s(w_{82})]$ is missing, because it is outside the current window, while its final segment $(s(w_{82}), f(i_2^{t,q_{81}})]$ has been extended, given the events that arrived after q_{81}. Considering the intervals i^s_1 and i^t_2 of Figure 2(a), it is not possible to compute meets(i^s_1, i^t_2) at q_{82} because i^s_1 and part of i^t_2 take place before w_{82}. To address this issue, we cache, at q_{81}, $i_1^{s,q_{81}}$ and the segment of $i_2^{t,q_{81}}$ that is before time-point $s(w_{82})$. Figure 3(b) depicts these cached (segments of) intervals with dotted lines. At q_{82}, $i_1^{s,q_{81}}$, which matches i^s_1, is added to $\mathcal{S}$ and the cached segment of $i_2^{t,q_{81}}$ is amalgamated with $i_2^{t,q_{82}}$, forming an interval that matches i^t_2. As a result, we compute that meets(i^s_1, i^t_2) holds and the output interval $i_1^{dia,q_{82}}$ at q_{82}. In Figure 3(b), the dashed segment of $i_1^{dia,q_{82}}$ denotes the interval part that falls outside w_{82}. In contrast to $i_1^{dia,q_{81}}$, $i_1^{dia,q_{82}}$ matches i_1^{dia}, i.e., the output interval displayed in Figure 2(a). □

Algorithm 3 *windowing*$(\mathcal{S}, \mathcal{T}, \text{rel})$

1: $s(w_{j+1}) = q_j + step - \omega$
2: $\mathcal{S}_< \leftarrow getIntervalsBeforeTimepoint(\mathcal{S}, s(w_{j+1}))$
3: $i^s_* \leftarrow getIntervalContainingTimepoint(\mathcal{S}, s(w_{j+1}))$
4: $i^t_* \leftarrow getIntervalContainingTimepoint(\mathcal{T}, s(w_{j+1}))$
5: **if** $i^s_* \neq null$ **then**
6:   **if** rel ∈ {meets, overlaps, before} **or** ($i^t_* \neq null$ **and** ((rel ∈ {starts, equal} **and** $s(i^s_*) = s(i^t_*)$) **or** (rel ∈ {finishes, during} **and** $s(i^s_*) > s(i^t_*)$))) **then**
7:     $cache([s(i^s_*), s(w_{j+1})])$
8: **if** $i^t_* \neq null$ **then**
9:   **if** rel ∈ {meets, starts, overlaps} **and** $\exists i^s \in \mathcal{S}_<$: rel(i^s, i^t_*) **then**
10:     $cache([s(i^t_*), s(w_{j+1})])$, $cache(i^s)$
11:   **else if** rel ∈ {before} **and** $\exists i^s \in \mathcal{S}_<$: (before(i^s, i^t_*) **and** $f(i^s) \geq s(w_{j+1}) - mem$) **then**
12:     $cache([s(i^t_*), s(w_{j+1})])$
13:   **else if** rel ∈ {finishes, during} **or** ($i^s_* \neq null$ **and** ((rel ∈ {starts, equal} **and** $s(i^s_*) = s(i^t_*)$) **or** (rel ∈ {overlaps} **and** $s(i^s_*) < s(i^t_*)$))) **then**
14:     $cache([s(i^t_*), s(w_{j+1})])$
15:   **if** rel ∈ {during} **then**
16:     **for** $i^s \in \mathcal{S}_<$: rel(i^s, i^t_*) **do** $cache(i^s)$
17: **if** rel ∈ {before} **then**
18:   **for** $i^s \in \mathcal{S}_< : f(i^s) \geq s(w_{j+1}) - mem$ **do** $cache(i^s)$

Example 5 demonstrates that the prefix of a target interval intersecting with the next window, and a source interval ending before the next window, may need to be cached to guarantee correct reasoning. Target intervals ending before the next window are not cached because they cannot satisfy any Allen relation with a source interval ending in the future. Algorithm 3 presents our caching procedure. First, we compute the start endpoint of the next window $s(w_{j+1})$ (line 1), and identify the list of source intervals $\mathcal{S}_<$ taking place before $s(w_{j+1})$, as well as the source and target intervals i^s_* and i^t_*, if any, containing $s(w_{j+1})$ (lines 2–4). For example, in Figure 3(a), $\mathcal{S}_< = [i_1^{s,q_{81}}]$, $i_*^s = null$ and $i_*^t = i_2^{t,q_{81}}$. The segments of i_*^s and i_*^t that are before $s(w_{j+1})$ may need to be cached (see $i_2^{t,q_{81}}$ in Example 5). The conditions for caching $[s(i_*^s), s(w_{j+1})]$ and $[s(i_*^t), s(w_{j+1})]$ are in lines 5–7 and 8–14, respectively. Moreover, we may need to cache a subset of the intervals in $\mathcal{S}_<$. The conditions for caching such intervals are presented in lines 8–10 and 15–18.

In the case of before, it is impossible to guarantee correct reasoning without caching every source interval. For example, interval $i_1^{s,q_{81}}$ of Figure 3(a) will satisfy before with all target intervals after $i_2^{t,q_{81}}$ that arrive in the future. Thus, we need to always keep $i_1^{s,q_{81}}$ in memory to ensure correctness. In order to maintain a balance between efficiency and correctness in the case of before, we use a memory threshold mem and cache, at query time q_j, all source intervals ending in $[s(w_{j+1}) - mem, s(w_{j+1})]$ (see lines 17–18 of Algorithm 3). If at least one of these intervals is before i_*^t, i.e., the target interval containing $s(w_{j+1})$, then we also cache $[s(i_*^t), s(w_{j+1})]$ (see lines 11–12). This way, we may compute, at q_{j+1}, the interval pairs (i^s, i^t) satisfying before, such that i^t intersects with window w_{j+1} and i^s ends in $[s(w_{j+1}) - mem, s(w_{j+1})]$.

3.3 Correctness and Complexity

We prove the correctness of RTEC$_\text{A}$ and present its complexity, with respect to Allen relation computation for CER. The corresponding analyses on CER without Allen relations may be found in (Mantenoglou et al. 2022; Artikis et al. 2015).

Proposition 2 (Correctness of RTEC$_\text{A}$)**.** RTEC$_\text{A}$ computes all maximal intervals of a statically determined fluent defined in terms of an Allen relation, and no other interval. ▲

As expected, RTEC$_\text{A}$ is correct provided that interval delays, if any, can be tolerated by the window size. In other words, all intervals occurring before query time q_j that were not available at q_j, take place after $s(w_k)$, where $k > j$, and will be available by query time q_k. For the case of before, we additionally permit delayed source intervals taking place in $[s(w_k) - mem, s(w_k)]$ and arriving by q_k.

To prove the correctness of RTEC$_\text{A}$, we first show that Algorithm 2 computes all interval pairs (i^s, i^t), where $i^s \in \mathcal{S}$ and $i^t \in \mathcal{T}$, satisfying an Allen relation, and no other interval pair. Then, we show that Algorithm 3 caches all intervals that may be required by Algorithm 2 in the future for correct Allen relation computation, and no other interval.

Lemma 1. Algorithm 2 computes all interval pairs (i^s, i^t), where $i^s \in \mathcal{S}$ and $i^t \in \mathcal{T}$, satisfying an Allen relation, and no other interval pair. △

Proof. We present the proof for meets; the proofs for the remaining relations are similar and may be found in the supplementary material. Algorithm 2 is sound because, according to line 9, it may only compute an interval pair (i^s, i^t), such that $i^s \in \mathcal{S}$ and $i^t \in \mathcal{T}$, if meets(i^s, i^t) holds. Towards proving completeness, suppose that meets(i^s, i^t) holds and Algorithm 2 does not compute (i^s, i^t). In this case, according to line 9, there is no iteration of the *while* loop of Algorithm 2 such that pointer p_s points to i^s and p_t points to i^t. The condition of line 2 states that Algorithm 2 iterates over all items in at least one of its input lists. Suppose that, in the current iteration, p_s points to i^s when p_t points to an interval i_b^t that is before i^t. By the definition of meets, we have $f(i^s) = s(i^t)$, while it holds that $s(i^t) > f(i_b^t)$, because $\mathcal{T}$ is a sorted list of maximal intervals. Therefore, it holds that $f(i^s) > f(i_b^t)$, and thus we only increment pointer p_t (see lines 10–13). This condition continues to hold for all target intevals until p_t points to i^t. Similarly, assume that p_t points to i^t when p_s points to an interval i_b^s that is before i^s. $s(i^t) = f(i^s) > f(i_b^s)$ holds, and thus Algorithm 2 increments p_s until it points to i^s. In both cases, we reach an iteration of the *while* loop where p_s points to i^s and p_t points to i^t, which is a contradiction. Thus, if meets(i^s, i^t) holds, then Algorithm 2 computes (i^s, i^t). □

Lemma 2. Algorithm 3 caches all intervals that may satisfy an Allen relation with an interval arriving in the future, and no other interval. △

Proof. Suppose that there is a source interval i_*^s containing the start of the next window $s(w_{j+1})$. We will prove that Algorithm 3 caches $[s(i_*^s), s(w_{j+1})]$ iff i_*^s may satisfy an Allen relation with a target interval i^t arriving in the future. meets/overlaps/before: if i^t occurs in the next window w_{j+1}, it holds that $s(i^t) > s(i_*^s)$, and thus i_*^s may satisfy meets, overlaps or before with i^t. Therefore, Algorithm 3 caches $[s(i_*^s), s(w_{j+1})]$ (line 6). starts/equal: starts(i_*^s, i^t) and equal(i_*^s, i^t) may hold only if $s(i_*^s) = s(i^t)$ and $f(i_*^s) \leq f(i^t)$, in which case i^t also contains $s(w_{j+1})$. Thus, we cache $[s(i_*^s), s(w_{j+1})]$ iff there is a target interval starting at $s(i_*^s)$ and containing $s(w_{j+1})$ (see the conditions for starts and equal in line 6). finishes/during: finishes(i_*^s, i^t) and during(i_*^s, i^t) may hold only if $s(i_*^s) > s(i^t)$ and $f(i_*^s) \leq f(i^t)$. Therefore, we cache $[s(i_*^s), s(w_{j+1})]$ iff there is a target interval starting before $s(i_*^s)$ and containing $s(w_{j+1})$ (see the conditions for finishes and during in line 6). The proofs for caching a target interval containing $s(w_{j+1})$ and source intervals ending before $s(w_{j+1})$ are provided in the supplementary material. □

Proposition 3 (Complexity of RTEC$_\text{A}$)**.** The cost of computing the maximal intervals of a statically determined fluent defined in terms of an Allen relation is $\mathcal{O}(n)$, where n is the number of input intervals. ▲

We identified the conditions according to which a source (resp. target) interval cannot satisfy an Allen relation with any future target (source) interval. Algorithm 2 leverages these conditions (lines 10 and 12) in order to compute all interval pairs satisfying an Allen relation in a *single pass* over the input intervals. Moreover, we specified the conditions that allow us to detect in *linear time* the intervals that need to be cached (see lines 5–18 of Algorithm 3). In practice, the number of cached intervals is negligible. Thus the cost of computing Allen relations remains constant as the stream progresses, and is bound by the size of the window.

4 Experimental Analysis

Experimental Setup. For our empirical analysis, we employed real data streams from the field of maritime situa-

Batch size	Reasoning Time		
Input Intervals	$RTEC_A$	AEGLE	D^2IA
200	**1**	980	2K
2K	**14**	4K	6K
20K	**154**	71.5K	395K
200K	**1.8K**	MEM	>3.6M

(a) Batch setting.

Window size		Reasoning Time		Output Interval Pairs	
Days	Input Intervals	$RTEC_A$	D^2IA	$RTEC_A$	D^2IA
1	125	**1**	48	5K	5K
2	250	**2**	164	19K	18K
4	500	**4**	568	72K	71K
8	1K	**8**	1.7K	237K	236K
16	2K	**15**	7.8K	878K	874K

(b) Streaming setting.

Window size		Reasoning Time		Output Intervals	
Days	Input Intervals	$RTEC_A$	D^2IA	$RTEC_A$	D^2IA
1	19K	**40**	410	6K	6K
2	37K	**65**	592	9K	9K
4	74K	**99**	1.1K	16K	16K
8	148K	**156**	1.6K	32K	31K
16	297K	**285**	2.7K	77K	76K

(c) CER with Allen relations.

Table 3: Average reasoning times for Allen relation computation ((a) and (b)) and CER (c) in milliseconds.

tional awareness. The input events were derived from Automatic Identification System (AIS) position signals emitted by vessels, including information about their heading, speed and navigational status. Upon a stream of such events, we detect dangerous, suspicious and illegal vessel activities, such as $disappearedInArea$ (see rule (6)). The complete event description is available with the source code of $RTEC_A$[1]. We employed a publicly available dataset[2] including 18M AIS position signals, emitted from 5K vessels sailing in the Atlantic Ocean around the post of Brest, France, between October 2015–March 2016.

We compared $RTEC_A$ with AEGLE and D^2IA. AEGLE (Georgala et al. 2016) is a state-of-the-art system computing Allen relations that has been used in the link discovery framework LIMES (Ngomo et al. 2021). AEGLE reduces each Allen relation into a subset of eight atomic comparisons between interval endpoints. Moreover, AEGLE caches the outcome of each atomic comparison in order to reuse it in future relation evaluations. D^2IA is a CER system extending the Big Data stream processing engine Flink with interval-based semantics (Awad et al. 2022). Furthermore, D^2IA includes operators for reasoning over durative events using Allen relations. $RTEC_A$ operated in SWI-8.4 Prolog, while AEGLE and D^2IA operated on Java OpenJDK 18. The experiments were run on a core of a desktop PC running Ubuntu 22.04, with AMD Ryzen 7 5700U CPU @ 1.8GHz and 16GB RAM. Our empirical analysis is reproducible; the code and the data of our experiments are publicly available[1].

Experimental Results. AEGLE does not support windowing or CER. Thus, the aim of the first set of experiments was to compare $RTEC_A$, AEGLE and D^2IA in a batch setting, for Allen relation computation without CER. We instructed these systems to derive all interval pairs satisfying an Allen relation among lists of maximal intervals during which composite maritime activities were detected on the Brest dataset. We evaluated the efficiency of each framework as the number of input intervals increases, while making sure that all systems produced the same interval pairs (see Lemma 1 for the correctness of $RTEC_A$ in a batch setting). As expected, the most common Allen relation was before, while relations requiring endpoing equality, i.e., meets, starts, finishes and equal, were less frequently satisfied. Table 3(a) shows the average reasoning times of $RTEC_A$, AEGLE and D^2IA for computing all Allen relations among input lists containing 200–200K intervals. All results displayed in Table 3 are the average of 30 experiments. Since AEGLE and D^2IA do not assume that the input lists are temporally sorted, the interval lists of composite maritime activities were not sorted. The performance of AEGLE and D^2IA on sorted input lists is almost identical to that presented in Table 3(a), and thus omitted here. For $RTEC_A$, we had to sort the input lists prior to Allen relation computation; the cost of sorting is included in the reported times of $RTEC_A$.

Table 3(a) shows that the reasoning time of $RTEC_A$ increases linearly with the input size, verifying our complexity analysis (see Proposition 3). Moreover, $RTEC_A$ outperforms AEGLE and D^2IA by 2–3 orders of magnitude. For example, in the experiments with 200K input intervals, $RTEC_A$ was able to compute all interval pairs satisfying an Allen relation in about 1.9 seconds. In contrast, AEGLE terminated with a memory error, while we killed the execution of D^2IA because it lasted for more than one hour. AEGLE sorted each input interval list by start or end endpoint, depending on the relation under evaluation. However, both sorting operations produce the same result on a list of maximal intervals, and thus only one of them is sufficient. $RTEC_A$ leverages the common assumption in CER that intervals are maximal, and avoids such unnecessary re-computations. D^2IA has higher reasoning times than $RTEC_A$ and AEGLE, because it is significantly slower when computing before, which is satisfied by most interval pairs in each experiment. $RTEC_A$ evaluates before very efficiently as it derives all target intervals satisfying before with some source interval in a single iteration. Furthermore, in contrast to D^2IA and AEGLE, $RTEC_A$ uses a compact representation for the computed interval pairs in order to avoid their explicit enumeration (see lines 6 and 8 of Algorithm 2).

In our next set of experiments, we compared $RTEC_A$ with D^2IA for Allen relation computation in a streaming setting, but without CER. D^2IA does not support overlapping windows for Allen relations and does not cache intervals that may satisfy an Allen relation, such as before, in the future. Thus, to facilitate a fair comparison, we set the step of $RTEC_A$ to the window size and the threshold mem to zero. Table 3(b) presents the average reasoning times of $RTEC_A$ and D^2IA, and the average number of interval pairs com-

[2]https://zenodo.org/record/1167595

puted by each system (see 'output interval pairs'). The input lists were provided to each system in windows, ranging from 1 day, including approx. 125 intervals, to 16 days, including 2K intervals. Our results show that RTEC_A remains orders of magnitude faster than D^2IA. Moreover, the cost of our caching mechanism is negligible (e.g., compare the second line of Table 3(a) with the last line of Table 3(b)), verifying our complexity analysis. Note that RTEC_A computed more interval pairs than D^2IA in most settings. This is due to the fact that D^2IA does not include a technique for transferring intervals to future windows (with the exception of open intervals), compromising correctness. See Lemma 2 for the correctness of RTEC_A in a streaming setting.

In the final set of experiments, we compared RTEC_A and D^2IA on CER, using fifteen patterns of composite maritime activities with Allen relations, such as those presented in Section 3.1. Given a pattern including $\mathsf{allen}(\mathsf{rel}, \mathcal{S}, \mathcal{T}, \mathsf{outMode}, I)$, RTEC_A computes all interval pairs of $\mathcal{S}$ and $\mathcal{T}$ satisfying rel, and applies outMode to the computed pairs, in order to produce the maximal intervals of the composite activity defined by the pattern. In contrast, D^2IA computes only the union of the interval pairs satisfying an Allen relation within a composite activity pattern, i.e., D^2IA does not allow the specification of another output mode. To facilitate a fair comparison, we set the outMode of RTEC_A to union in all maritime patterns.

Table 3(c) presents the average reasoning times of RTEC_A and D^2IA, and the average number of composite activity intervals (see 'output intervals'). The number of input intervals is significantly larger as compared to our previous experiments, because the input intervals correspond to activities performed by *all* vessels in the dataset. In CER, we are interested in the combinations of input items indicated by the composite activity patterns, and not on evaluating all possible interval combinations, as in the previous experiments. Consequently, the number of composite activity intervals is much smaller than the number of input intervals. Our results show that RTEC_A is significantly faster than D^2IA, without compromising correctness (in some cases D^2IA misses composite activities due to the absence of interval caching).

5 Summary, Related and Further Work

We proposed RTEC_A, a CER system supporting Allen relations in temporal patterns. We presented the syntax, semantics and reasoning algorithms of RTEC_A, proved its correctness, and showed that it has linear complexity bound by the window size. Moreover, we compared RTEC_A with AEGLE and D^2IA, two state-of-the-art computational frameworks supporting Allen relations, on real maritime data, demonstrating the benefits of RTEC_A.

Several approaches in the literature are related to our work. CORE (Bucchi et al. 2022) is an automata-based CER engine deriving durative composite events efficiently, using a compact data structure for maintaining partial matches. However, relations in the language of CORE can only be unary. Thus, CORE cannot express, e.g., the maritime patterns of RTEC_A. A comparison of automata-based and logic-based CER systems may be found in the recent survey of Giatrakos et al. (2020). LARS (Beck et al. 2018) is a formal stream reasoning language that can express interval-based rules. LARS-based reasoners (Urbani et al. 2022; Eiter et al. 2019; Beck et al. 2017; Bazoobandi et al. 2017), however, support only a fragment of LARS that cannot express interval derivations. s(CASP) (Arias et al. 2022) is a query-driven execution model for Answer Set Programming with constraints, supporting Event Calculus-based reasoning. jREC is an implementation of the Event Calculus, using caching and indexing techniques for interval-based CER (Falcionelli et al. 2019). None of these systems supports Allen relations. Moreover, RTEC has proven very efficient (in real applications), outperforming related systems, such as jREC (Mantenoglou et al. 2022).

Several CER systems do support Allen relations. TPStream (Körber et al. 2019) transforms instantaneous events into durative *situations*, and computes temporal patterns, including Allen relations, over situation intervals. In TPStream, situations cannot be defined in terms of multiple event types or background knowledge. For example, it is not possible to express $withinArea$ (see rules (1)–(3)). Moreover, the cost of Allen relation computation in TPStream is $\mathcal{O}(nlogn)$, where n is the number of input intervals, which is higher than that of RTEC_A. ISEQ (Li et al. 2011) processes streams of durative events and allows for Allen relations. Unlike RTEC_A, neither ISEQ nor TPStream supports relational patterns, which is a significant limitation for CER. Furthermore, ISEQ does not allow for the derivation of intervals from instantaneous events or the specification of an output interval when an Allen relation is satisfied. ETALIS (Anicic et al. 2012) is an event-driven stream reasoning system that supports Allen relations. Similar to AEGLE, ETALIS and ISEQ do not take advantage of the common assumption in CER that activity intervals are maximal, compromising performance.

Havelund et al. (2021) proposed an extension of Allen's algebra featuring quantification over intervals. This work focuses on relations before, during and overlaps, omitting the remaining relations. Unlike RTEC_A, this approach does not support the construction of intervals by means of (arbitrary conditions on) concurrent events. nfer (Kauffman et al. 2018) is a rule-based system transforming instantaneous event streams into interval-based, hierarchical abstractions, possibly using Allen relations. nfer does not include optimisations for Allen relation computation and does not guarantee correct reasoning in a streaming setting. Several approaches compute Allen relations using versions of the plane-sweeping algorithm. For example, Piatov et al. (2021) developed a family of interval join algorithms, including Allen relations, with log-linear time complexity. Chekol et al. (2019) extended SPARQL with plane-sweeping-based algorithms for Allen relation computation; however, this work does not support streaming data. None of the aforementioned frameworks for Allen relation computation is designed to handle the inherent delays in streams.

TPStream and the system of Chawda et al. (2014) support distributed Allen relation computation. Pilourdault et al. (2016) compute approximate incarnations of Allen relations. Extending RTEC_A with approximate Allen relations and distribution techniques are future work directions.

Acknowledgements

We would like to thank Christos Doulkeridis for his comments at the early stages of this work. This work was supported by the EU-funded CREXDATA project (No 101092749), and by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd and the 4th Call for HFRI PhD Fellowships (Fellowship Numbers: 6011 and 10860).

References

Allen, J. 1984. Towards a general theory of action and time. *Artif. Intell.* 23(2):123–154.

Anicic, D.; Rudolph, S.; Fodor, P.; and Stojanovic, N. 2012. Stream reasoning and complex event processing in ETALIS. *Semantic Web* 3(4):397–407.

Arias, J.; Carro, M.; Chen, Z.; and Gupta, G. 2022. Modeling and reasoning in event calculus using goal-directed constraint answer set programming. *Theory Pract. Log. Program.* 22(1):51–80.

Artikis, A.; Sergot, M. J.; and Paliouras, G. 2015. An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng.* 27(4):895–908.

Awad, A.; Tommasini, R.; Langhi, S.; Kamel, M.; Valle, E. D.; and Sakr, S. 2022. D^2ia: User-defined interval analytics on distributed streams. *Inf. Syst.* 104:101679.

Bazoobandi, H. R.; Beck, H.; and Urbani, J. 2017. Expressive stream reasoning with laser. In *ISWC*, volume 10587, 87–103.

Beck, H.; Eiter, T.; and Folie, C. 2017. Ticker: A system for incremental asp-based stream reasoning. *Theory Pract. Log. Program.* 17(5-6):744–763.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2018. LARS: A logic-based framework for analytic reasoning over streams. *Artif. Intell.* 261:16–70.

Brendel, W.; Fern, A.; and Todorovic, S. 2011. Probabilistic event logic for interval-based event recognition. In *CVPR*, 3329–3336.

Bucchi, M.; Grez, A.; Quintana, A.; Riveros, C.; and Vansummeren, S. 2022. CORE: a complex event recognition engine. *Proc. VLDB Endow.* 15(9):1951–1964.

Chawda, B.; Gupta, H.; Negi, S.; Faruquie, T. A.; Subramaniam, L. V.; and Mohania, M. K. 2014. Processing interval joins on map-reduce. In *EDBT*, 463–474.

Chekol, M. W.; Pirrò, G.; and Stuckenschmidt, H. 2019. Fast interval joins for temporal SPARQL queries. In *Companion of WWW*, 1148–1154. ACM.

Cugola, G., and Margara, A. 2012. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.* 44(3):15:1–15:62.

Eiter, T.; Ogris, P.; and Schekotihin, K. 2019. A distributed approach to LARS stream reasoning (system paper). *Theory Pract. Log. Program.* 19(5-6):974–989.

Falcionelli, N.; Sernani, P.; de la Torre, A. B.; Mekuria, D. N.; Calvaresi, D.; Schumacher, M.; Dragoni, A. F.; and Bromuri, S. 2019. Indexing the event calculus: Towards practical human-readable personal health systems. *Artif. Intell. Medicine* 96:154–166.

Georgala, K.; Sherif, M. A.; and Ngomo, A. N. 2016. An efficient approach for the generation of allen relations. In *ECAI*, volume 285, 948–956.

Giatrakos, N.; Alevizos, E.; Artikis, A.; Deligiannakis, A.; and Garofalakis, M. N. 2020. Complex event recognition in the big data era: a survey. *VLDB J.* 29(1):313–352.

Havelund, K.; Omer, M.; and Peled, D. 2021. Monitoring first-order interval logic. In *SEFM*, volume 13085, 66–83. Springer.

Kauffman, S.; Havelund, K.; Joshi, R.; and Fischmeister, S. 2018. Inferring event stream abstractions. *Formal Methods Syst. Des.* 53(1):54–82.

Körber, M.; Glombiewski, N.; Morgen, A.; and Seeger, B. 2019. Tpstream: low-latency and high-throughput temporal pattern matching on event streams. *Distributed and Parallel Databases* 39:361–412.

Kowalski, R., and Sergot, M. 1986. A logic-based calculus of events. *New Gen. Comput.* 4(1).

Li, M.; Mani, M.; Rundensteiner, E. A.; and Lin, T. 2011. Complex event pattern detection over streams with interval-based temporal semantics. In *DEBS*, 291–302. ACM.

Mantenoglou, P.; Pitsikalis, M.; and Artikis, A. 2022. Stream reasoning with cycles. In *KR*, 544–553.

Ngomo, A. N.; Sherif, M. A.; Georgala, K.; Hassan, M. M.; Dreßler, K.; Lyko, K.; Obraczka, D.; and Soru, T. 2021. LIMES: A framework for link discovery on the semantic web. *Künstliche Intell.* 35(3):413–423.

Piatov, D.; Helmer, S.; Dignös, A.; and Persia, F. 2021. Cache-efficient sweeping-based interval joins for extended allen relation predicates. *VLDB J.* 30(3):379–402.

Pilourdault, J.; Leroy, V.; and Amer-Yahia, S. 2016. Distributed evaluation of top-k temporal joins. In *SIGMOD*, 1027–1039. ACM.

Pitsikalis, M.; Artikis, A.; Dreo, R.; Ray, C.; Camossi, E.; and Jousselme, A. 2019. Composite event recognition for maritime monitoring. In *DEBS*, 163–174. ACM.

Santipantakis, G. M.; Vlachou, A.; Doulkeridis, C.; Artikis, A.; Kontopoulos, I.; and Vouros, G. A. 2018. A stream reasoning system for maritime monitoring. In *TIME*, volume 120, 20:1–20:17.

Song, Y. C.; Kautz, H. A.; Allen, J. F.; Swift, M. D.; Li, Y.; Luo, J.; and Zhang, C. 2013. A markov logic framework for recognizing complex events from multimodal data. In *ICMI*, 141–148.

Tsilionis, E.; Artikis, A.; and Paliouras, G. 2022. Incremental event calculus for run-time reasoning. *J. Artif. Intell. Res.* 73:967–1023.

Urbani, J.; Krötzsch, M.; and Eiter, T. 2022. Chasing streams with existential rules. In *KR*.