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. The language of RTEC includes events and fluent-value pairs (FVP) holds continuously during the maximal intervals in list $I$.

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.
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 happensAt\((E, T)\), expressing a stream of event instances.
- Rules with head initiatedAt\((F = V, T)\) or terminatedAt\((F = V, T)\), expressing the effects of events on FVPs.
- Rules with head 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{align*}
\text{initiatedAt}(\text{withinArea}\(Vl, \text{AreaType}\) = true, T) & \leftarrow \text{happensAt}(\text{enters}\(Vl, \text{AreaID}\), T), \text{areaType}(\text{AreaID} , \text{AreaType}). \\
\text{terminatedAt}(\text{withinArea}\(Vl, \text{AreaType}\) = true, T) & \leftarrow \text{happensAt}(\text{leaves}\(Vl, \text{AreaID}\), T), \text{areaType}(\text{AreaID} , \text{AreaType}). \\
\text{terminatedAt}(\text{withinArea}\(Vl, \text{AreaType}\) = true, T) & \leftarrow \text{happensAt}(\text{gapStart}(Vl), T).
\end{align*}
\]

\(\text{withinArea}\(Vl, \text{AreaType}\)\) is a Boolean simple fluent denoting that a vessel \(Vl\) is in some area of \(\text{AreaType}\), while \(\text{enters}\(Vl, \text{AreaID}\)\), \(\text{leaves}\(Vl, \text{AreaID}\)\) and \(\text{gapStart}(Vl)\) are input events, derived by the online processing of vessel position signals, and their spatial relations with areas of interest (Santipantakos et al. 2018). \(\text{areaType}(\text{AreaID} , \text{AreaType})\) is an atemporal predicate storing background knowledge concerning the types of areas in a dataset. Rules (1) and (2) state that \(\text{withinArea}\(Vl, \text{AreaType}\)\) is initiated (resp. terminated) as soon as vessel \(Vl\) enters (leaves) an area \(\text{AreaID}\), whose type is \(\text{AreaType}\). According to rule (3), \(\text{withinArea}\(Vl, \text{AreaType}\)\) 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, \(\text{holdsFor}(\text{withinArea}\(Vl, \text{AreaType}\) = true, I)\), i.e., the list of maximal intervals \(I\) during which \(Vl\) is in \(\text{AreaType}\).

The syntax of simple fluent definitions is provided in the supplementary document. In addition to the domain-independent definition of \(\text{holdsFor}\), which is used for computing the maximal intervals of simple fluents, an event description may include domain-specific \(\text{holdsFor}\) 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: \(\text{union}\), \(\text{intersect}\), \(\text{relative_complement}\), \(\text{intervalConstruct}\) of a FVP other than \(F\).

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

\[
\begin{align*}
\text{ holdsFor}(\text{anchoredOrMoored}(Vl) = true, I) & \leftarrow \\
\text{ holdsFor}(\text{stopped}(Vl) = \text{farFromPorts}, I_{sf}), \\
\text{ holdsFor}(\text{stopped}(Vl) = \text{nearPorts}, I_{sn}), \\
\text{ union}(I_{sf}, I_{sn}), I.
\end{align*}
\]

\(\text{anchoredOrMoored}(Vl)\) is a Boolean statically determined fluent defined in terms of the FVPs \(\text{stopped}(Vl)\) and \(\text{withinArea}\(Vl, \text{anchorage}\) = true\). \(\text{stopped}(Vl)\) is a multi-valued fluent expressing the periods during which
Table 1 presents relations and pairwise disjoint relations among intervals (Allen 1984). Allen’s interval algebra specifies thirteen jointly exhaustive relations, 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.

### 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 point of interval $i$, respectively.

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 $i^s \in S$ and $i^t \in T$, satisfying before, intersect, union, complement and complement inv. Table 2 shows the possible values of outMode and Output list $I$.

### Allen Relations in Event Descriptions

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

According to $\text{Allen}(\text{rel}, S, T, \text{outMode}, I)$, $I$ contains the maximal intervals produced by applying outMode to the interval pairs of the ‘source list’ $S$ and the ‘target list’ $T$ 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 $S_{\text{rel}} = \{i^s | i^s \in S \land \exists i^t \in T : \text{rel}(i^s, i^t)\}$ and $T_{\text{rel}} = \{i^t | i^t \in T \land \exists i^s \in S : \text{rel}(i^s, i^t)\}$, 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

<table>
<thead>
<tr>
<th>Relation</th>
<th>Definition</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>before$(i^s, i^t)$</td>
<td>$f(i^s) &lt; s(i^t)$</td>
<td>$i^s$ $i^t$</td>
</tr>
<tr>
<td>meets$(i^s, i^t)$</td>
<td>$f(i^s) = s(i^t)$</td>
<td>$i^s$ $i^t$</td>
</tr>
<tr>
<td>starts$(i^s, i^t)$</td>
<td>$s(i^s) = s(i^t)$, $f(i^s) &lt; f(i^t)$</td>
<td>$i^s$</td>
</tr>
<tr>
<td>finishes$(i^s, i^t)$</td>
<td>$s(i^s) &gt; s(i^t)$, $f(i^s) = f(i^t)$</td>
<td>$i^s$</td>
</tr>
<tr>
<td>during$(i^s, i^t)$</td>
<td>$s(i^s) &gt; s(i^t)$, $f(i^s) &lt; f(i^t)$</td>
<td>$i^s$</td>
</tr>
<tr>
<td>overlaps$(i^s, i^t)$</td>
<td>$s(i^s) &lt; s(i^t)$, $f(i^s) &gt; s(i^t)$, $f(i^s) &lt; f(i^t)$</td>
<td>$i^s$</td>
</tr>
<tr>
<td>equal$(i^s, i^t)$</td>
<td>$s(i^s) = s(i^t)$, $f(i^s) = f(i^t)$</td>
<td>$i^s$</td>
</tr>
</tbody>
</table>

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

\[
\text{holdsFor} \left( \text{disappearedInArea} \left( V_l, \text{AreaType} \right) = \text{true}, I_{\text{dia}} \right) \leftarrow \\
\text{holdsFor} \left( \text{withinArea} \left( V_l, \text{AreaType} \right) = \text{true}, S \right), \\
\text{holdsFor} \left( \text{gap} \left( V_l \right) = \text{farFromPorts}, T \right), \\
\text{allen} \left( \text{meets}, S, T, \text{target}, I_{\text{dia}} \right). \\
\]

(6)

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

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

\[
\text{holdsFor} \left( \text{suspiciousRendezVous} \left( V_{l1}, V_{l2} \right) = \text{true}, I_{\text{srv}} \right) \leftarrow \\
\text{holdsFor} \left( \text{gap} \left( V_{l1} \right) = \text{farFromPorts}, I_{g_1} \right), \\
\text{holdsFor} \left( \text{gap} \left( V_{l2} \right) = \text{farFromPorts}, I_{g_2} \right), \\
\text{holdsFor} \left( \text{proximity} \left( V_{l1}, V_{l2} \right) = \text{true}, T \right), \\
\text{union_all} \left( \left[ I_{g_1}, I_{g_2}, S \right], \right), \text{allen} \left( \text{during}, \left[ S, T, \text{target}, I_{\text{srv}} \right] \right). \\
\]

(7)

\text{suspiciousRendezVous} \left( V_{l1}, V_{l2} \right) \) is a statically determined fluent, and \( \text{proximity} \left( V_{l1}, V_{l2} \right) \) is a Boolean fluent denoting whether two vessels, \( V_{l1} \) and \( V_{l2} \), are close to each other. \( T \), 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). \( \text{union_all} \) in rule (7) derives \( S \), i.e., the list of maximal intervals during which \( \text{gap} \left( V_{l1} \right) = \text{farFromPorts} \) or \( \text{gap} \left( V_{l2} \right) = \text{farFromPorts} \). Then, \( \text{allen} \left( \text{during}, \left[ S, T, \text{target}, I_{\text{srv}} \right] \right) \) identifies the maximal intervals of \( T \) that contain an interval of \( S \) and stores them in list \( I_{\text{srv}} \). Therefore, rule (7) specifies that vessels \( V_{l1} \) and \( V_{l2} \) may be conducting an illegal activity during an interval \( i_{\text{srv}} \), if \( i_{\text{srv}} \) is an interval during which \( V_{l1} \) and \( V_{l2} \) are close to each other, and \( i_{\text{srv}} \) contains an interval \( i^* \) during which at least one of the vessels stops transmitting its position. Figure 2(b) displays an illustration of rule (7), \( \text{union_all} \) constructs list \( S \) as the union of the intervals in lists \( I_{g_1} \) and \( I_{g_2} \). Among the intervals of \( S \) and \( T \), during is only satisfied for the interval pair \( \left( i^t_2, i^t_1 \right) \), resulting in \( I_{\text{srv}} = \left[ i^t_2 \right] \). Note that \( I_{\text{srv}} \) cannot be derived by the interval manipulation constructs of RTEC. \( \text{union_all} \left( \left[ S, T, \text{target}, I \right] \right) \), e.g., would compute list \( I = \left[ i^t_1, i^t_2, i^t_3 \right] \), including intervals \( i^t_1 \) and \( i^t_3 \) that do not satisfy during.

As mentioned earlier, Table 2 lists the possible values of \text{outMode} and their meaning. Consider the computation of \( \text{allen} \left( \text{meets}, S, T, \text{outMode}, I_{\text{dia}} \right) \) in the example of Figure 2(a), where the only interval pair satisfying meets is \( \left( i^t_1, i^t_2 \right) \). If \text{outMode} was complement or complement_inv, we would apply the \text{relative_complement_all} construct on \( i^t_1 \) and \( i^t_2 \) (complement_inv reverses the order of operants in \text{relative_complement_all} and compute, respectively, list \( I_{\text{dia}} \) as \( \left[ i^t_2 \setminus i^t_1 \right] \) or \( \left[ i^t_1 \setminus i^t_2 \right] \)). Note that some combinations of rel and \text{outMode} are equivalent. For instance, if an interval pair \( \left( i^t, i^t \right) \) satisfies during, then \( i^t \) is a subinterval of \( i^t \). Therefore, \( \text{allen} \left( \text{during}, S, T, \text{union}, I \right) \) and \( \text{allen} \left( \text{during}, S, T, \text{target}, I \right) \) produce the same output list.

Semantics. An event description in RTEC 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 \( \left( F_j = V_j, F_i = V_i \right) \) iff:
   - There is an initiatedAt or terminatedAt rule for \( F_j = V_j \) having \text{holdsAt} \left( F_j = V_j, T \right) as one of its conditions.
   - There is a \text{holdsFor} rule for \( F_i = V_i \) having \text{holdsFor} \left( F_i = V_i, \right) as one of its conditions.

According to Definition 4, the addition of allen constructs
in stochastically 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). An event description in RTEC is a locally stratified logic program. ▲

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

### 3.2 Reasoning

We extend the notion of stochastically determined fluent event of RTEC with algorithms computing Allen relations. Algorithm 1 presents the steps of the evaluation of $\text{allen}(\text{rel, } S, T, \text{outMode, } I)$. This algorithm derives all interval pairs $(i^s, i^t)$, such that $i^s \in S$ and $i^t \in T$, satisfying Allen relation $\text{rel}$, and stores them in list $\text{pairs}$ (line 3). We then compute lists $S_{\text{rel}}$ and $T_{\text{rel}}$ containing, respectively, the source and the target intervals appearing in list $\text{pairs}$ at least once (line 4). This way, we may apply outMode to lists $S_{\text{rel}}$ and $T_{\text{rel}}$, 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 $S$ and $T$ 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 $S$ and $T$ satisfying an Allen relation $\text{rel}$ and stores them in list $\text{pairs}$. $I$ in 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, $S$ and $T$ are also sorted lists of maximal intervals (see Definition 3). We evaluate $\text{rel}$ by means of interval endpoint comparisons, following the corresponding definition in Table 1. An element of $\text{pairs}$ is a tuple of the form $(i^s, T')$, denoting that the source interval $i^s$ satisfies $\text{rel}$ with every interval in the list of target intervals $T' \subseteq T$. Using this compact representation, we avoid enumerating all computed interval pairs, without information loss. For example, the tuple $(i^s_1, [i^t_3, i^t_4])$ for a relation $\text{rel}$ denotes that $\text{rel}(i^s_1, i^t_3)$ and $\text{rel}(i^s_1, i^t_4)$ hold.

Algorithm 2 uses two pointers, $p_s$ and $p_t$, to traverse $S$ and $T$. If $\text{rel}$ is before and, indeed, before $(i^s, i^t)$ holds, we add the tuple $(i^s, [i^t_3, i^t_4], T' \text{length}(T))$ to $\text{pairs}$, where $T' \text{length}(T)$ denotes the last interval of $T$ (line 6). If 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 $(i^s, [T[p_s+1], \ldots, T' \text{length}(T)])$ to $\text{pairs}$ (line 8). If $\text{rel}$ is not before and $\text{rel}(i^s, i^t)$ holds, we simply add $(i^s, i^t)$ to $\text{pairs}$ (line 9).

Afterwards, Algorithm 2 increments pointer $p_s$ or/and pointer $p_t$. $p_s$ (resp. $p_t$) may be incremented only if the current source (target) interval $i^s$ ($i^t$) cannot satisfy $\text{rel}$ with any subsequent target (source) interval. Since $S$ and $T$ 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 $\text{rel}$, without iterating over any subsequent interval of $S$ and $T$. The conditions in which pointers $p_s$ and $p_t$ may be incremented, while guaranteeing the correct computation of interval pairs satisfying $\text{rel}$, are presented in lines 10 and 12.

#### Example 4 (Allen relation computation). In the example of Figure 2(a), $\text{Allen}(\text{meets, } S, T, \text{target, } I_{\text{dis}})$ is used to compute the maximal intervals of $\text{disappearedInArea}$ in list $I_{\text{dis}}$. In order to derive these intervals, RTEC computes all interval pairs in lists $S$ and $T$ satisfying $\text{meets}$ (see line 3 of Algorithm 1). This is achieved with Algorithm 2. In this example, the source list is $S = [i^s_1, i^s_2]$, the target list is $T = [i^t_1, i^t_2]$. Initially, $p_s$ points to $i^s_1$ and $p_t$ points to $i^t_2$. Algorithm 2 verifies that $\text{meets}$ does not hold for the interval pair $(i^s_1, i^t_2)$ in line 9. Next, we check whether pointer $p_s$ needs to be incremented. Since $f(i^s_1) > f(i^t_2)$, 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 $p_s$ (line 13). The interval pair of the next iteration is $(i^s_2, i^t_2)$, $\text{meets}(i^s_2, i^t_2)$ is satisfied and thus we compute the pair $(i^s_2, [i^t_1, i^t_2])$. $i^s_2$ and $i^t_2$ cannot satisfy $\text{meets}$ 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 $(i^s_2, [i^t_1, i^t_2])$. □

#### Windowing

In order to handle streaming data, CER systems often employ windowing techniques. At each ‘query time’ $q_t$, RTEC reasons over the items of an input stream that fall within a specified sliding window $w_j = (q_j - \omega, q_j)$, where $\omega$ 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 $\omega$ and not on the complete stream. The size of $\omega$ and the temporal distance between two consecutive query times, i.e., the ‘step’ $q_{j+1} - q_j$, may be manually set or optim-
misled 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 \( \omega \) 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 \( S \) and \( T \) of Figure 2(a) as they are available at query times \( q_{S1} \) and \( q_{S2} \). The corresponding windows \( w_{S1} \) and \( w_{S2} \) are overlapping in order to accommodate, at query time \( q_{S2} \), stream elements that took place in \((q_{S2} - \omega, q_{S1}] \), but arrived after \( q_{S1} \).

RTEC\(_A\) 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\(_A\) guarantees correct reasoning by computing, at \( q_j \), all interval pairs \((i^*, i^j)\) satisfying Allen relation rel, such that at least one of \( i^* \) and \( i^j \) 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 \( \text{all-meets}(S, T, \text{target}, I_{\text{dia}}) \) at query time \( q_{S1} \), where \( S \) and \( T \) contain only the intervals that fall within window \( w_{S1} \). Contrast these intervals with the ones depicted in Figure 2(a). Interval \( i^1_{q_{S1}} \), e.g., is shorter than the interval \( i^j_2 \) of Figure 2(a) because a segment of \( i^j_2 \) falls outside \( w_{S1} \). Moreover, the events leading to the extension of \( i^1_{q_{S1}} \) up to \( q_{S1} \) have been delayed and are not available at \( q_{S1} \), and thus \( i^1_{q_{S1}} \) ends earlier than \( q_{S1} \). Based on the intervals in \( w_{S1} \), we compute that \( \text{meets}(i^j_2, i^1_{q_{S1}}) \) holds at \( q_{S1} \) and derive the output interval \( i^1_{\text{dia}, q_{S1}} \), which matches \( i^1_{q_{S1}} \).

The intervals of \( S \) and \( T \) available at the next query time, \( q_{S2} \), i.e., \( i^2_{q_{S1}} \) and \( i^j_{q_{S2}} \), are displayed in Figure 3(b). The first segment of \( i^2_{q_{S2}} \), i.e., \([s(i^2_{q_{S1}}), s(w_{S2})]\) is missing, because it is outside the current window, while its final segment \([s(w_{S2}), f(i^2_{q_{S2}})]\) has been extended, given the events that arrived after \( q_{S1} \). Considering the intervals \( i^j_2 \) and \( i^2_2 \) of Figure 2(a), if it is not possible to compute \( \text{meets}(i^j_2, i^2_2) \) at \( q_{S2} \) because \( i^j_2 \) and part of \( i^2_2 \) take place before \( w_{S2} \). To address this issue, we cache, at \( q_{S1} \), \( i^j_2 \) and the segment of \( i^2_2 \) that is before time-point \( s(w_{S2}) \). Figure 3(b) depicts these cached (segments of) intervals with dotted lines. At \( q_{S2} \), \( i^2_{q_{S2}} \), which matches \( i^j_2 \), is added to \( S \) and the cached segment of \( i^2_{q_{S2}} \) is amalgamated with \( i^2_{q_{S2}} \), forming an interval that matches \( i^2_2 \). As a result, we compute that \( \text{meets}(i^j_2, i^2_2) \) holds and the output interval \( i^2_{\text{dia}, q_{S2}} \) at \( q_{S2} \).

In Figure 3(b), the dashed segment of \( i^2_{\text{dia}, q_{S2}} \) denotes the interval part that falls outside \( w_{S2} \). In contrast to \( i^i_{\text{dia}, q_{S1}} \), \( i^j_{\text{dia}, q_{S2}} \) matches \( i^j_{\text{dia}} \), i.e., the output interval displayed in Figure 2(a).

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 \( S_c \) taking place before \( s(w_{j+1}) \), as well as the source and target intervals \( i^s \) and \( i^j \), if any, containing \( s(w_{j+1}) \) (lines 2–4). For example,
Lemma 1. Allen relation computation, and no other interval. Then, we show that Algorithm 3 caches all intervals and not available at \( q \) defined in terms of an Allen relation, and no other interval.

\( □ \)

(Correctness of RTEC found in (Mantenoglou et al. 2022; Artikis et al. 2015).

In order to maintain a balance between efficiency and correctness, with respect to Allen relation computation for CER. The algorithm 3). If at least one of these intervals is before \( q \), the interval pairs \((i^*, i^*)\) satisfying before, such that \( i^* \) intersects with window \( w_{j+1} \) and \( i^* \) ends in \([s(w_{j+1}) - mem, s(w_{j+1})] \).

3.3 Correctness and Complexity

We prove the correctness of RTECA 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 RTECA). RTECA computes all maximal intervals of a statically determined fluent defined in terms of an Allen relation, and no other interval.

As expected, RTECA 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 RTECA, we first show that Algorithm 2 computes all interval pairs \((i^*, i^*)\), where \( i^* \in S \) and \( i^* \in T \), 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 \((i^*, i^*)\), where \( i^* \in S \) and \( i^* \in T \), 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 \((i^*, i^*)\), such that \( i^* \in S \) and \( i^* \in T \), if \( \text{meets}(i^*, i^*) \) holds. Towards proving completeness, suppose that \( \text{meets}(i^*, i^*) \) holds and Algorithm 2 does not compute \((i^*, i^*)\). 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 \( i^* \) and \( p_t \) points to \( i^* \). 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 \( i^* \) when \( p_t \) points to an interval \( i^* \) that is before \( i^* \). By the definition of \( \text{meets} \), we have \( f(i^*) > f(i^*) \), while it holds that \( f(i^*) > f(i^*) \), because \( T \) is a sorted list of maximal intervals. Therefore, it holds that \( f(i^*) > f(i^*) \), and thus we only increment pointer \( p_t \) (see lines 10–13). This condition continues to hold for all target intervals until \( p_s \) points to \( i^* \). Similarly, assume that \( p_s \) points to \( i^* \) when \( p_s \) points to an interval \( i^* \) that is before \( i^* \). \( s(i^*) = f(i^*) > f(i^*) \) holds, and thus Algorithm 2 increments \( p_s \) until it points to \( i^* \). In both cases, we reach an iteration of the while loop where \( p_s \) points to \( i^* \) and \( p_s \) points to \( i^* \), which is a contradiction. Thus, if \( \text{meets}(i^*, i^*) \) holds, then Algorithm 2 computes \((i^*, i^*)\).

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

Proposition 3 (Complexity of RTECA). The cost of computing the maximal intervals of a statically determined fluent defined in terms of an Allen relation is \( O(n) \), 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 \( \text{linear time} \) 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-
and 16GB RAM. Our empirical analysis is reproducible; the Ubuntu 22.04, with AMD Ryzen 7 5700U CPU @ 1.8GHz experiments were run on a core of a desktop PC running while AEGLE and D\textsuperscript{2}IA using Allen relations. RTEC\textsuperscript{IA} includes operators for reasoning over durative events (Awad et al. 2022). Furthermore, AEGLE caches the outcome of each atomic comparison in order to reuse it in future relation evaluations. D\textsuperscript{2}IA does not support wins, but without CER. D\textsuperscript{2}IA does 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\textsuperscript{2}IA on sorted input lists is almost identical to that presented in Table 3(a), and thus omitted here. For RTEC\textsuperscript{IA}, we had to sort the input lists prior to Allen relation computation; the cost of sorting is included in the reported times of RTEC\textsuperscript{IA}.

We compared RTEC\textsuperscript{IA} with AEGLE and D\textsuperscript{2}IA. 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\textsuperscript{2}IA is a CER system extending the Big Data stream processing engine Flink with interval-based semantics (Awad et al. 2022). Furthermore, D\textsuperscript{2}IA includes operators for reasoning over durable events using Allen relations. RTEC\textsuperscript{IA} operated in SWI-8.4 Prolog, while AEGLE and D\textsuperscript{2}IA 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\textsuperscript{3}.}

### Experimental Results

AEGLE does not support windowing or CER. Thus, the aim of the first set of experiments was to compare RTEC\textsuperscript{IA}, AEGLE and D\textsuperscript{2}IA 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\textsuperscript{IA} in a batch setting). As expected, the most common Allen relation was before, while relations requiring endpoint equality, i.e., meets, starts, finishes and equal, were less frequently satisfied. Table 3(a) shows the average reasoning times of RTEC\textsuperscript{IA}, AEGLE and D\textsuperscript{2}IA 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\textsuperscript{2}IA 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\textsuperscript{2}IA on sorted input lists is almost identical to that presented in Table 3(a), and thus omitted here. For RTEC\textsuperscript{IA}, we had to sort the input lists prior to Allen relation computation; the cost of sorting is included in the reported times of RTEC\textsuperscript{IA}.

Table 3(a) shows that the reasoning time of RTEC\textsuperscript{IA} increases linearly with the input size, verifying our complexity analysis (see Proposition 3). Moreover, RTEC\textsuperscript{IA} outperforms AEGLE and D\textsuperscript{2}IA by 2–3 orders of magnitude. For example, in the experiments with 200K input intervals, RTEC\textsuperscript{IA} 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\textsuperscript{2}IA 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\textsuperscript{IA} leverages the common assumption in CER that intervals are maximal, and avoids such unnecessary re-computations. D\textsuperscript{2}IA has higher reasoning times than RTEC\textsuperscript{IA} and AEGLE, because it is significantly slower when computing before, which is satisfied by most interval pairs in each experiment. RTEC\textsuperscript{IA} 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\textsuperscript{2}IA and AEGLE, RTEC\textsuperscript{IA} 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\textsuperscript{IA} with D\textsuperscript{2}IA for Allen relation computation in a streaming setting, but without CER. D\textsuperscript{2}IA 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\textsuperscript{IA} to the window size and the threshold mem to zero. Table 3(b) presents the average reasoning times of RTEC\textsuperscript{IA} and D\textsuperscript{2}IA, and the average number of interval pairs com-

---

\textsuperscript{3}https://zenodo.org/record/1167595

<table>
<thead>
<tr>
<th>Batch size</th>
<th>Reasoning Time</th>
<th>Output Interval Pairs</th>
<th>Window size</th>
<th>Reasoning Time</th>
<th>Output Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Intervals</td>
<td>RTEC\textsuperscript{IA}</td>
<td>AEGLE</td>
<td>D\textsuperscript{2}IA</td>
<td>Days</td>
<td>Input Intervals</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>980</td>
<td>2K</td>
<td>1</td>
<td>125</td>
</tr>
<tr>
<td>2K</td>
<td>14</td>
<td>4K</td>
<td>6K</td>
<td>2</td>
<td>250</td>
</tr>
<tr>
<td>20K</td>
<td>154</td>
<td>71.5K</td>
<td>395K</td>
<td>4</td>
<td>500</td>
</tr>
<tr>
<td>200K</td>
<td>1.8K</td>
<td>MEM &gt; 3.6M</td>
<td></td>
<td>8</td>
<td>1K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>2K</td>
</tr>
</tbody>
</table>

(a) Batch setting. (b) Streaming setting. (c) CER with Allen relations.

Table 3: Average reasoning times for Allen relation computation ((a) and (b)) and CER (c) in milliseconds.
ties performed by experiments, because the input intervals correspond to activities is significantly larger as compared to our previous experiments. Note that RTEC\textsubscript{A} computed more interval pairs than D\textsuperscript{2}\textsubscript{IA} in most settings. This is due to the fact that D\textsuperscript{2}\textsubscript{IA} 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\textsubscript{A} in a streaming setting.

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

Table 3(c) presents the average reasoning times of RTEC\textsubscript{A} and D\textsuperscript{2}\textsubscript{IA}, and the average number of composite activity intervals (see ‘output interval pairs’). 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\textsubscript{A} is significantly faster than D\textsuperscript{2}\textsubscript{IA}, without compromising correctness (in some cases D\textsuperscript{2}\textsubscript{IA} misses composite activities due to the absence of interval caching).

5 Summary, Related and Further Work

We proposed RTEC\textsubscript{A}, a CER system supporting Allen relations in temporal patterns. We presented the syntax, semantics and reasoning algorithms of RTEC\textsubscript{A}, proved its correctness, and showed that it has linear complexity bound by the window size. Moreover, we compared RTEC\textsubscript{A} with AEGLE and D\textsuperscript{2}\textsubscript{IA}, two state-of-the-art computational frameworks supporting Allen relations, on real maritime data, demonstrating the benefits of RTEC\textsubscript{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\textsubscript{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 (Mantenegro 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 $O(n\log n)$, where $n$ is the number of input intervals, which is higher than that of RTEC\textsubscript{A}. ISEQ (Li et al. 2011) processes streams of durative events and allows for Allen relations. Unlike RTEC\textsubscript{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\textsubscript{A}, this approach does not support the construction of intervals by means of (arbitrary conditions on) concurrent events. \texttt{ infr} (Kauffmann et al. 2018) is a rule-based system transforming instantaneous event streams into interval-based, hierarchical abstractions, possibly using Allen relations. \texttt{ infr} does not include optimizations 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\textsubscript{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