Sequencing in the Run-Time Event Calculus

Periklis Mantenoglou[a] **and Alexander Artikis**[b,c]

[a]Örebro University, Sweden, [b]NCSR "Demokritos", Greece, [c]University of Piraeus, Greece
periklis.mantenoglou@oru.se, a.artikis@iit.demokritos.gr

Abstract. Composite event recognition (CER) systems detect instances of composite activities over streams of timestamped events. A fundamental operator for CER is 'sequencing', expressing that two activities take place one after the other. There is no consensus on a universal definition for sequencing. We provide a set of required properties for a sequencing operator for CER, i.e., an interval-based semantics, required for durative activities, and associativity, required to express activity hierarchies. We propose a sequencing operator that satisfies all requirements, as opposed to the ones in the literature, and we implement our operator in the CER engine RTEC. We compare our operator both theoretically and empirically with state-of-the-art approaches, demonstrating its benefits and limitations.

1 Introduction

Complex event recognition (CER) systems process high-velocity streams in order to detect instances of (spatio-)temporal pattern satisfaction with minimal latency [12]. 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 [31].

CER systems are typically automata-based [17, 18, 30, 23]—see Giatrakos et al. [17] for a survey. CORE and Wayeb, e.g., are automata-based frameworks that have proven highly efficient compared to the state-of-the-art [9, 1]. While automata-based systems commonly include a *sequencing operator*, expressing phenomena where activities take place one after the other, they do not always agree on its intended semantics. For instance, several frameworks express sequencing only over instantaneous activities, and fail to support durative phenomena, which are common in CER [6, 21]. On the other hand, frameworks that do support interval-based sequencing often do so using an operator that is not associative, leading to semantic ambiguities when used in hierarchical patterns [16, 27, 35].

Logic-based CER frameworks support features that are not typically found in automata-based approaches, such as relational and hierarchical patterns with background knowledge [8, 33, 34]. These approaches, however, do not support inertial activities. The Event Calculus is a logic programming formalism for reasoning about the effects of events over time [22]. It features a built-in representation of the law inertia, allowing the specification of the initiations and the terminations of durative activities, which may persist over time. Though several implementations of the Event Calculus have been proposed [11, 29, 10, 19, 7, 3, 26, 15], the Run-Time Event Calculus (RTEC[1]) has proven to be the most effective at reasoning over large data streams and complex temporal specifications [24, 25].

RTEC, however, does not support sequencing. While there is an extension of RTEC that supports Allen's interval algebra [25], this extension does not capture an associative sequencing operator. To tackle this issue, we propose RTEC_S, an extension of RTEC that support sequencing via an associative, interval-based operator, addressing the semantic issues raised about sequencing in CER [35].

Our contributions may be summarised as follows. First, we outline a set of requirements for sequencing in CER, i.e., maximal, disjoint interval representation and associativity. Second, we propose a sequencing operator that, contrary to the ones in the literature, fulfills these requirements. Third, we propose RTEC_S, an extension of RTEC that supports sequencing via our operator. RTEC_S is the only CER system that captures both inertial and sequential phenomena. We present the syntax, the semantics, and the reasoning algorithm of RTEC_S, along with a discussion on correctness, complexity, and accuracy under windowing. The proofs of all propositions are provided in the supplementary material[2]. Fourth, we compare the sequencing operator of RTEC_S with the ones found in state-of-the-art CER frameworks, such as CORE and Wayeb. Fifth, we present a reproducible empirical evaluation of RTEC_S on an artificial and a real domain, including a comparison with a state-of-the-art system.

2 Background: RTEC

RTEC is a formal, logic programming framework that extends the Event Calculus with optimisation techniques for CER [4, 24, 25].

Syntax. The language of RTEC includes sorts for representing time, instantaneous events and fluents, i.e., properties whose values may change over time. RTEC employs a linear time-line with non-negative integer time-points. A 'fluent-value pair' (FVP) $F{=}V$ denotes that fluent F has value V. In CER, FVPs are used to express the composite activities that we are interested in detecting. $\mathsf{happensAt}(E, T)$ signifies that event E occurs at time-point T. $\mathsf{initiatedAt}(F = V, T)$ (resp. $\mathsf{terminatedAt}(F = V, T)$) expresses that a time period during which a fluent F has the value V continuously is initiated (terminated) at T. $\mathsf{holdsAt}(F = V, T)$ states that F has value V at T, while $\mathsf{holdsFor}(F = V, I)$ expresses that $F{=}V$ holds continuously in the intervals included in list I.

A formalisation of the activity definitions of a domain in RTEC is called *event description*. An event description may contain rules defining two types of FVPs: 'simple' and 'statically determined'. A simple FVP is defined using a set of initiatedAt and terminatedAt rules, and is subject to the commonsense law of inertia, i.e., an FVP $F{=}V$ holds at a time-point T, if $F{=}V$ has been 'initiated' by an

[1] github.com/aartikis/rtec

[2] https://periklismant.github.io/appendices/ecai25.pdf

event at a time-point earlier than T, and not 'terminated' by another event in the meantime.

Example 1 (Within area)**.** In maritime monitoring, an activity may be disallowed in certain areas, e.g., fisheries restricted areas. Thus, it is desirable to compute the intervals during which a vessel is in such an area. See the definition of a simple FVP below:

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

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

$\mathit{withinArea}(\mathit{Vl}, \mathit{AreaType})$ is a Boolean fluent denoting that a vessel Vl is in an area of type $\mathit{AreaType}$, while $\mathit{entersArea}(\mathit{Vl}, \mathit{AreaID})$ and $\mathit{leavesArea}(\mathit{Vl}, \mathit{AreaID})$, derived by the online processing of vessel position signals, and their spatial relations with areas of interest. $\mathit{areaType}(\mathit{AreaID}, \mathit{AreaType})$ is an atemporal predicate storing background knowledge regarding the types of areas in a dataset. Rules (1) and (2) state that $\mathit{withinArea}(\mathit{Vl}, \mathit{AreaType})$ is initiated (resp. terminated) as soon as vessel Vl enters (leaves) an area AreaID with type $\mathit{AreaType}$. ♢

The syntax of the rules defining simple FVPs is presented in [24].

A statically determined FVP $F = V$ is defined via a rule with head holdsFor$(F = V, I)$. This rule computes the maximal, disjoint intervals (MDIs) during which $F = V$ holds continuously by applying a set interval manipulation operations, i.e., union_all, intersect_all and relative_complement_all, on the MDIs of other FVPs.

Example 2 (Anchored and moored vessels)**.** Consider the following definition of a statically determined FVP:

$$\begin{aligned}&\mathsf{holdsFor}(\mathit{anchoredOrMoored}(\mathit{Vl}) = \mathsf{true}, I) \leftarrow \\ &\quad \mathsf{holdsFor}(\mathit{stopped}(\mathit{Vl}) = \mathit{farFromPorts}, I_{sf}), \\ &\quad \mathsf{holdsFor}(\mathit{withinArea}(\mathit{Vl}, \mathit{anchorage}) = \mathsf{true}, I_a), \\ &\quad \mathsf{intersect\_all}([I_{sf}, I_a], I_{sfa}), \\ &\quad \mathsf{holdsFor}(\mathit{stopped}(\mathit{Vl}) = \mathit{nearPorts}, I_{sn}), \\ &\quad \mathsf{union\_all}([I_{sfa}, I_{sn}], I).\end{aligned} \tag{3}$$

$\mathit{anchoredOrMoored}(\mathit{Vl})$ is a Boolean statically determined fluent, defined in terms of three other FVPs: $\mathit{stopped}(\mathit{Vl}) = \mathit{farFromPorts}$, $\mathit{stopped}(\mathit{Vl}) = \mathit{nearPorts}$ and $\mathit{withinArea}(\mathit{Vl}, \mathit{anchorage}) = \mathsf{true}$. The multi-valued fluent $\mathit{stopped}(\mathit{Vl})$ expresses the periods during which vessel Vl is idle near some port or far from all ports. Rule (3) 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 MDIs 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 MDIs during which vessel Vl has stopped near some port or within an anchorage area. ♢

Definition 1 (Syntax of Rules Defining Statically Determined FVPs)**.** The definition of statically determined FVP $F = V$ is a rule that has the following syntax:

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

The first body literal of a holdsFor rule defining $F = V$ is a holdsFor predicate expressing the MDIs of an FVP other than $F = V$. This is followed by a possibly empty list, denoted by '[[ ]]', of holdsFor predicates and interval manipulation constructs, expressed by intervalConstruct. intervalConstruct(L_j, I_{n+j}) may be one of the following: union_all(L_j, I_{n+j}), intersect_all(L_j, I_{n+j}) or relative_complement_all(I_k, L_j, I_{n+j}). I_k, where $k < n + j$, is a list of MDIs appearing earlier in the body of the rule, and list L_j contains a subset of these lists. The output list I_{n+m} contains the MDIs during which $F = V$ holds continuously. ■

Semantics. An event description defines a *dependency graph* expressing the relationships between the FVPs of the event description.

Definition 2 (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 holdsAt$(F_j = V_j, T)$ as one of its conditions.
   - There is a holdsFor rule for $F_i = V_i$ having holdsFor$(F_j = V_j, I)$ as one of its conditions. ■

A stratification of the FVPs of an event description may be constructed by following the edges of the dependency graph bottom-up, leading to the following result [25].

Proposition 1 (Semantics of RTEC)**.** An event description in RTEC is a locally stratified logic program [32]. ♦

Reasoning. The key reasoning task of RTEC is the computation of holdsFor$(F = V, I)$, i.e., the list of MDIs I during which a FVP expressing a composite activity holds continuously. For a simple FVP $F = V$, RTEC first computes the initiations and the terminations of $F = V$, by evaluating its initiatedAt and its terminatedAt rules, respectively. Next, RTEC computes the MDIs of $F = V$ by matching each initiation T_s of $F{=}V$ with the first termination T_e of $F{=}V$ after T_s, ignoring every intermediate initiation between T_s and T_e. RTEC may then derive holdsAt$(F = V, T)$ by checking whether T belongs to one of the MDIs of $F{=}V$. In the case of a statically determined FVP $F = V$, RTEC computes holdsFor$(F = V, I)$ by evaluating the conditions of the holdsFor rule with FVP $F = V$ in its head.

RTEC supports hierarchical event descriptions, where it is possible to compute and cache the MDIs of FVPs in a bottom-up manner, guided by the corresponding dependency graph [4]. This way, the intervals of an FVP are computed and cached at most once, and are retrieved from memory when required in the definitions of other FVPs, thus avoiding re-computations.

3 Sequencing

In CER, it is common for a composite activity to be defined as a sequence of other activities. In banking, e.g., a sequence of transactions of the same credit card in distant locations may indicate fraud [5]. In cybersecurity, adversary tactics may be composed of several sequential steps, such as content injection and privilege escalation [2]. In maritime monitoring, a fishing trip may be defined as a sequence of (i) being moored in some port, (ii) entering a fishing area, (iii) fishing, (iv) exiting the fishing area, (v) returning to the port. As a result, activity specification formalisms require a *sequencing operator* ";", which receives as input two activities α_1 and α_2 and constructs another activity $\alpha_1 ; \alpha_2$, expressing the sequence of α_1 and α_2.

Given an activity α and an input activity stream S, we use $[[\alpha]]_S$ to denote the occurrences of α given stream S. α may be an item of the input stream S or a composite activity whose occurrences are derived via temporal pattern matching over the items in S. For an input activity α, we adopt the common assumption in CER that the occurrences of α are non-overlapping.

Assumption 1 (Stream of Activity MDIs)**.** Let S be a stream and α an activity, where α is an item of S. $[[\alpha]]_S$ is composed of MDIs. ✠

Note that the intervals in $[[\alpha]]_S$ may be instantaneous.

We identify a set of requirements that a sequencing operator should meet, in order to be suitable for CER. We start with the well-documented need of interval-based semantics [35, 28].

Requirement 1 (Interval-based Semantics for Sequencing)**.** Consider a sequencing operator ";", a stream S and activities α_1 and α_2. $[[\alpha_1\, ; \alpha_2]]_S$ is composed of MDIs. ◇

Associating MDIs with activities is both a prerequisite for supporting RTEC and a reasonable choice for CER due to the corresponding complexity gains (these will be demonstrated in later sections). A sequencing operator ";" that satisfies Requirement 1 is compositional, because an activity $\alpha_1\, ; \alpha_2$, constructed using ";", is represented with same type of time-stamp, i.e., MDIs, as its building block activities α_1 and α_2. As a result, $\alpha_1\, ; \alpha_2$ may be used as a building block in a different activity definition that includes ";". In other words, a sequencing operator that fulfills Requirement 1 has the following property.

Property 1 (Compositionality)**.** Consider two activities α_1 and α_1 and an input stream S. Sequencing is compositional iff $[[\alpha_1\, ; \alpha_2]]_S$, i.e., the occurrences of activity $\alpha_1\, ; \alpha_2$ based on stream S, have the same type as both $[[\alpha_1]]_S$ and $[[\alpha_2]]_S$. ⧫

Compositionality is necessary for activity hierarchies, i.e., activities being defined in terms of other non-input activities, which are common in CER [17].

In the presence of large hierarchies, activity definitions may be composed via sequencing several other activities. Moreover, an activity that is used as a building block for some definition may itself be defined as the sequence of other activities. Consider, e.g., an activity α_{123}, which is defined as the sequence of activities α_1, α_2 and α_3 There are two options for specifying α_{123}—we can either compose α_{123} as $\alpha_{12}\, ; \alpha_3$, where α_{12} is defined as $\alpha_1\, ; \alpha_2$, or as $\alpha_1\, ; \alpha_{23}$, where α_{23} is defined as $\alpha_2\, ; \alpha_3$. Deciding which is the preferred representation may depend on complexity concerns, which in turn may depend on the frequency of α_1, α_2 and α_3 [20, 14]. We need to ensure correctness regardless of which of the two options is chosen by the activity definition developer, i.e., both options should lead to the same intervals for α_{123}. In other words, sequencing in CER should be implemented using an *associative* operator.

Requirement 2 (Associativity)**.** Given a stream S and activities α_1, α_2 and α_3, $[[(\alpha_1 ; \alpha_2) ; \alpha_3]]_S$ is equal to $[[\alpha_1 ; (\alpha_2 ; \alpha_3)]]_S$. ◇

Notice that associativity implies compositionality.

The example below illustrates a problematic behavior of a non-associative sequencing operator [16, 35].

Example 3 (Non-Associative Sequencing)**.** Consider an activity α_{123}, which is defined as the sequence of activities α_1, α_2 and α_3, and a sequencing operator ";" such that, for every pair of activities α_a and α_b and stream S, we have $[[\alpha_a ; \alpha_b]]_S = [[\alpha_b]]_S$.

Let S be a stream such that $[[\alpha_1]]_S = \{(5, 7)\}$, $[[\alpha_2]]_S = \{(1, 3)\}$ and $[[\alpha_3]]_S = \{(9, 11)\}$. Based on the sequencing operator we employ in this example, we have $[[\alpha_1\, ; \alpha_2]]_S = \emptyset$ and $[[\alpha_2\, ; \alpha_3]]_S = \{(9, 11)\}$. Therefore, we may compute $[[\alpha_{123}]]_S$ as $[[(\alpha_1\, ; \alpha_2)\, ; \alpha_3]]_S = \emptyset$ or $[[\alpha_1\, ;(\alpha_2\, ; \alpha_3)]]_S = \{(9, 11)\}$. ◊

The behavior exemplified above is undesirable; based on the intervals associated with activities α_1, α_2 and α_3, according to which α_2 occurs earlier than α_1, α_{123} does not take place, and the order of applying the sequencing operator should not alter this result.

An associative sequencing operator allows for optimisations [35]. Consider again the example of activity α_{123}, and a stream S where activity α_1 occurs infrequently. In this case, the preferred option is to define α_{123} as $(\alpha_1\, ; \alpha_2)\, ; \alpha_3$, and not as $\alpha_1\, ;(\alpha_2\, ; \alpha_3)$. Computing first $[[\alpha_2\, ; \alpha_3]]_S$ for each sequence of α_2 and α_3 to only then reject the match on the grounds that there is no instance of α_1 may lead to redundant computations. Instead, starting with the computation of $[[\alpha_1\, ; \alpha_2]]_S$ should improve efficiency as α_1 appears infrequently, and thus fewer potential matches for α_{123} are considered.

4 A Naive Approach to Sequencing under MDIs

Our goal is to define an activity sequencing operator ";" that abides by Requirements 1 and 2. Following [35], we start with an *abstract sequencing model*, which includes the main components of ";".

Definition 3 (Abstract Sequencing Model)**.** An abstract sequencing model is a tuple $(T, \prec, \mathsf{S}, \otimes)$, where

- T is a set of time-stamps.
- $\prec$ is a partial order on T.
- $\mathsf{S} : T \times 2^T \rightarrow 2^T$ is a *successor function*. It receives as input a time-stamp t and a set of candidate time-stamps $\mathcal{F}$ and returns a subset of $\mathcal{F}$, i.e., a set of immediate successor time-stamps for t.
- $\otimes : T \times T \rightarrow T$ is a *composition operator*. Given $t_1, t_2 \in T$, $t_1 \otimes t_2$ is the time-stamp of the sequence of two activities with time-stamps t_1 and t_2. ■

Based on the abstract sequencing model of Definition 3, it is possible to specify a concrete sequencing model by providing definitions for T, $\prec$, S and $\otimes$. The proposal of [35] for a concrete sequencing model is the one incorporated in the Cayuga framework [13]. We outline this model below, using $s(i)$ and $e(i)$ to denote the starting point and the ending point of an interval i.

Definition 4 (Sequencing Model of Cayuga)**.** The sequencing model of Cayuga is $(T_{cy}, \prec_{cy}, \mathsf{S}_{cy}, \otimes_{cy})$, where

- T_{cy} is the set of all intervals defined over the positive integers.
- for $i_1, i_2 \in T_{cy}$, $i_1 \prec_{cy} i_2$ iff $e(i_1) < s(i_2)$.
- $\mathsf{S}_{cy}(i_1, I_2) = \{i_2 \in I_2 \mid i_1 \prec i_2 \wedge \nexists i_2' \in I_2 : i_1 \prec_{cy} i_2' \wedge e(i_2') < e(i_2)\}$.
- $\forall i_1, i_2 \in T_{cy}$ such that $i_1 \prec_{cy} i_2$, we have $i_1 \otimes_{cy} i_2 = i$, where $s(i) = s(i_1)$ and $e(i) = e(i_2)$. ■

Based on the above sequencing model, we define the sequencing operator $[[.]]_S^{cy}$ of Cayuga over some stream S. If α is an item of S, we have $[[\alpha]]_S^{cy} = [[\alpha]]_S$. For a sequencing activity, we follow the definition below:

Definition 5 (Sequencing Operator in Cayuga)**.** Consider two activities α_1 and α_2 and a stream S. We have:

$$[[\alpha_1\, ; \alpha_2]]_S^{cy} = \{i_1 \otimes_{cy} i_2 | i_1 \in [[\alpha_1]]_S^{cy} \wedge i_2 \in \mathsf{S}_{cy}(i_1, [[\alpha_2]]_S^{cy})\} \quad ■$$

Unfortunately, this sequencing operator cannot be incorporated in RTEC because it does not abide by Requirement 1; for a stream S and activities α_1 and α_2, it is possible for $[[\alpha_1\, ; \alpha_2]]_S^{cy}$ to be composed of intervals that are not MDIs. Consider the following example.

Example 4. Consider activities α_1 and α_2, and a stream S such that $[[\alpha_1]]_S = \{i_{11}, i_{12}\}$, where $i_{11} = (1, 3)$ and $i_{12} = (5, 6)$, and $[[\alpha_2]]_S = \{i_2\}$, where $i_2 = (9, 11)$. Following Definition 4, we have $i_{11} \prec_{cy} i_2$, $i_{12} \prec_{cy} i_2$, $\mathsf{S}_{cy}(i_{11}, \{i_2\}) = \{i_2\}$, $\mathsf{S}_{cy}(i_{12}, \{i_2\}) = \{i_2\}$, $i_{11} \otimes_{cy} i_2 = (1, 11)$ and $i_{12} \otimes_{cy} i_2 = (5, 11)$. Therefore, according to Definition 5, the sequencing operator of

Cayuga computes the following intervals for $\alpha_1 ; \alpha_2$:

$$[[\alpha_1 ; \alpha_2]]_S^{cy} = \{(1, 11), (5, 11)\} \qquad \Diamond$$

Example 4 illustrates that, starting from two activities occurring in lists of MDIs, Cayuga may construct overlapping intervals for the sequence of these two activities.

White et al. [35] present a set of desired axioms for a sequencing model, which are fulfilled by Cayuga, and prove that there is no interval-based sequencing model that satisfies these axioms and is also associative. These axioms concern the more general case where the intervals of an activity may be overlapping, as opposed to being MDIs. Below, we define an associative sequencing operator for activities that take place in MDIs, thus fulfilling Requirements 1–2.

5 Sequencing in RTEC

We propose a sequencing operator that fulfills Requirements 1 and 2 and may be used in RTEC. Our operator functions under the assumption that the activities participating in sequencing are mutually exclusive, which is a common assumption in CER. In maritime situational awareness, e.g., the sequence of activities constituting a fishing trip—being moored, leaving a port, etc.—are mutually exclusive.

Assumption 2 (Mutually Exclusive Activities)**.** Consider two activities α_1 and α_2. If there is a pattern combining α_1 and α_2 using a sequencing operator, then α_1 and α_2 are mutually exclusive, i.e., for every stream S, $[[\alpha_1]]_S \cap [[\alpha_2]]_S = \emptyset$. ✠

Towards a sequencing operator for RTEC, we define a new type of successor function, compared to the one in Definition 3. Based on this new successor function, each interval is assigned at most one successor and at most one predecessor. This is motivated by the issue of Example 4, where, in the computation of $\alpha_1 ; \alpha_2$, the two intervals of α_1 are assigned the same successor interval, leading to overlapping intervals for $\alpha_1 ; \alpha_2$. We may avoid this by designating each interval of α_2 as the successor of at most one interval of α_1, i.e., each interval of α_2 has at most one predecessor. We pair 'adjacent' activity intervals as follows:

Definition 6 (Adjacency Mapping)**.** Consider a stream S and two activities α_1 and α_2. We define the adjacency mapping of an interval $i_1 \in [[\alpha_1]]_S$ as follows:

$$\mathsf{A}(i_1, [[\alpha_1]]_S, [[\alpha_2]]_S) = \begin{cases} i_2 & \text{if } \exists i_2 \in [[\alpha_2]]_S\text{: } i_1 \prec_{rt} i_2 \wedge \\ & (\neg \exists i_2' \in [[\alpha_2]]_S\text{: } i_1 \prec_{rt} i_2' \prec_{rt} i_2) \wedge \\ & (\neg \exists i_1' \in [[\alpha_1]]_S\text{: } i_1 \prec_{rt} i_1' \prec_{rt} i_2) \\ \emptyset & \text{otherwise} \end{cases} \qquad \blacksquare$$

According to Definition 6, interval $i_1 \in [[\alpha_1]]_S^{rt}$ is adjacent with interval $i_2 \in [[\alpha_2]]_S^{rt}$ if i_1 ends before the start of i_2, and there is no interval in $[[\alpha_1]]_S^{rt}$ or $[[\alpha_2]]_S^{rt}$ that is situated between i_1 and i_2.

Now we may define a sequencing model for RTEC. This model is based on the abstract sequencing model of Definition 3, with the exception of employing the adjacency mapping of Definition 6 instead of a weaker successor function.

Definition 7 (Temporal Sequencing Model for RTEC)**.** The temporal sequencing model of RTEC is $(T_{rt}, \prec_{rt}, \mathsf{A}, \otimes_{rt})$, where

- T_{rt} is a set of intervals over the positive integers.
- $\prec_{rt}$ is a partial order on T_{rt}, such that, for $i_1, i_2 \in T_{rt}$, we have $i_1 \prec i_2$ iff $e(i_1) < s(i_2)$.
- $\mathsf{A} : T_{rt} \times 2^{T_{rt}} \times 2^{T_{rt}} \rightarrow T_{rt}$ is the mapping in Definition 6.
- $\forall i_1, i_2 \in T_{rt}$: $i_1 \prec_{rt} i_2$, we have $i_1 \otimes_{rt} i_2 = i$, where $s(i) = s(i_1)$ and $e(i) = e(i_2)$. ■

Based on the temporal sequencing model of Definition 7, we define a sequencing operator for RTEC.

Definition 8 (Sequencing Operator for RTEC)**.** Consider two activities α_1 and α_2 and a stream S. We have:

$$[[\alpha_1 ; \alpha_2]]_S^{rt} = \{i_1 \otimes_{rt} i_2 \mid i_1 \in [[\alpha_1]]_S^{rt} \wedge \\ i_2 = \mathsf{A}(i_1, [[\alpha_1]]_S^{rt}, [[\alpha_2]]_S^{rt}) \wedge i_2 \neq \emptyset\} \qquad \blacksquare$$

The sequencing operator in Definition 8 may not lead to overlapping intervals. Below, we illustrate this for the input intervals over which Cayuga generated overlapping intervals (see Example 4).

Example 5. Consider activities α_1 and α_2, and a stream S such that $[[\alpha_1]]_S = \{i_{11}, i_{12}\}$, where $i_{11} = (1, 3)$ and $i_{12} = (5, 6)$, and $[[\alpha_2]]_S = \{i_2\}$, where $i_2 = (9, 11)$. Following Definition 7, we have $i_{11} \prec_{rt} i_2$, $i_{12} \prec_{rt} i_2$, $\mathsf{A}(i_{11}, \{i_{11}, i_{12}\}, \{i_2\}) = \emptyset$, $\mathsf{A}(i_{12}, \{i_{11}, i_{12}\}, \{i_2\}) = i_2$, and $i_{12} \otimes_{cy} i_2 = (5, 11)$. Therefore, according to Definition 8, we have $[[\alpha_1 ; \alpha_2]]_S^{rt} = \{(5, 11)\}$. ◊

Proposition 2 ($[[\alpha_1 ; \alpha_2]]_S^{rt}$ consists of MDIs)**.** Consider a stream S and activities α_1 and α_2. The intervals in $[[\alpha_1 ; \alpha_2]]_S^{rt}$ are MDIs. ♦

Proposition 2 shows that our sequencing operator (Definition 8) fulfills Requirement 1. This implies that our sequencing operator is compositional (see Property 1).

Next, we study associativity (Requirement 2).

Proposition 3 ($[[(\alpha_1 ; \alpha_2) ; \alpha_3]]_S^{rt} = [[\alpha_1 ;(\alpha_2 ; \alpha_3)]]_S^{rt}$)**.** Consider a stream S and activities α_1, α_2 and α_3. It holds that

$$i \in [[(\alpha_1 ; \alpha_2) ; \alpha_3]]_S^{rt} \text{ iff } i \in [[\alpha_1 ;(\alpha_2 ; \alpha_3)]]_S^{rt} \qquad \blacklozenge$$

Based on Propositions 2 and 3, our sequencing operator is indeed compatible with RTEC, and it is associative, thus avoiding the corresponding correctness issues (see, e.g., Example 3), and paving the way for reasoning optimisations.

6 Comparative Study

We present a comparison of the sequencing operator of RTEC with operators that are used in well-known CER approaches.

6.1 Sequencing with Allen relations

RTEC$_A$ is a extension of RTEC that supports the relations of Allen's interval algebra in composite activity definitions [25]. We focus on the implementation of the before relation in RTEC$_A$, which may be viewed as a form of sequencing. Given two activities α_1 and α_2, RTEC$_A$ determines the intervals in $\mathsf{before}(\alpha_1, \alpha_2)$ in two steps. First, for each interval i_1 of α_1 and interval i_2 of α_2, RTEC$_A$ checks whether interval pair (i_1, i_2) satisfies before, i.e., whether i_1 and i_2 satisfy condition $e(i_1) < s(i_2)$. Subsequently, for the set of interval pairs satisfying before, RTEC$_A$ employs an "output mode", in order to compute MDIs during which $\mathsf{before}(\alpha_1, \alpha_2)$ takes place. RTEC$_A$ supports three output modes: "source", selecting interval i_1 of an interval pair (i_1, i_2) for inclusion in $\mathsf{before}(\alpha_1, \alpha_2)$, "target", selecting interval i_2, and "union", selecting both intervals i_1 and i_2.

In order to formulate before as a sequencing operator, we use the following temporal sequencing model:

Definition 9 (Temporal Sequencing Model for RTEC$_A$)**.** The temporal sequencing model for RTEC$_A$ is $(T_a, \prec_a, \mathsf{S}_a, \otimes_a)$, where

- T_a is a set of intervals over the positive integers.
- $\prec_a$ is a partial order on T_a, such that, for $i_1, i_2 \in T_a$, we have $i_1 \prec_a i_2$ iff $e(i_1) < s(i_2)$.

- $\mathsf{S}_a(i_1, I_2) = \{i_2 \in I_2 \mid i_1 \prec_a i_2\}$.
- $\forall i_1, i_2 \in T_a$: $i_1 \prec_a i_2$, we have: (i) $i_1 \otimes_a i_2 = i_1$ if output mode is source, (ii) $i_1 \otimes_a i_2 = i_2$ if output mode is target, or (iii) $i_1 \otimes_a i_2 = i_1 \cup i_2$ if output mode is union. ■

before may be defined as a sequencing operator as follows:

Definition 10 (Sequencing Operator in RTEC$_\mathrm{A}$)**.** Consider two activities α_1 and α_2 and a stream S. We have:

$$[[\alpha_1\,;\alpha_2]]^a_S = \{i_1 \otimes_a i_2 \mid i_1 \in [[\alpha_1]]^a_S \wedge i_2 \in \mathsf{S}_a(i_1, [[\alpha_2]]^a_S)\} \quad ■$$

The sequencing operator of RTEC$_\mathrm{A}$ is not associative (Requirement 2). Consider, e.g., activities α_1, α_2 and α_3. For output mode source and a stream S such that $[[\alpha_1]]^a_S = \{(1,3)\}$, $[[\alpha_2]]^a_S = \{(9,11)\}$ and $[[\alpha_3]]^a_S = \{(5,7)\}$, we have $[[(\alpha_1\,;\alpha_2)\,;\alpha_3]]^a_S = \{(1,3)\}$ and $[[\alpha_1\,;(\alpha_2\,;\alpha_3)]]^a_S = \emptyset$. For output mode target and a stream S such that $[[\alpha_1]]^a_S = \{(5,7)\}$, $[[\alpha_2]]^a_S = \{(1,3)\}$ and $[[\alpha_3]]^a_S = \{(9,11)\}$, we have $[[(\alpha_1\,;\alpha_2)\,;\alpha_3]]^a_S = \emptyset$ and $[[\alpha_1\,;(\alpha_2\,;\alpha_3)]]^a_S = \{(9,11)\}$. For output mode union and a stream S such that $[[\alpha_1]]^a_S = \{(1,3)\}$, $[[\alpha_2]]^a_S = \{(9,11)\}$ and $[[\alpha_3]]^a_S = \{(5,7)\}$, we have $[[(\alpha_1\,;\alpha_2)\,;\alpha_3]]^a_S = \{(1,3),(5,7)\}$ and $[[\alpha_1\,;(\alpha_2\,;\alpha_3)]]^a_S = \emptyset$. In contrast, our sequencing operator is associative.

6.2 *Sequencing in Automata-based CER*

CORE is an automata-based CER engine in which a composite activity may be constructed by sequencing an arbitrary number of activities from the input stream. Given input activities α_1 and α_2, CORE computes that $\alpha_1\,;\alpha_2$ holds in an interval $(s(i_1), e(i_2))$ if α_1 occurs in i_1, α_2 occurs in i_2 and i_1 ends before the start of i_2. CORE defines the following temporal sequencing model:

Definition 11 (Temporal Sequencing Model for CORE)**.** The temporal sequencing model for CORE is $(T_{co}, \prec_{co}, \mathsf{S}_{co}, \otimes_{co})$, where

- T_{co} is the set of all intervals defined over the positive integers.
- for $i_1, i_2 \in T_{co}$, $i_1 \prec_{co} i_2$ iff $e(i_1) < s(i_2)$.
- $\mathsf{S}_{co}(i_1, I_2) = \{i_2 \in I_2 \mid i_1 \prec_{co} i_2\}$.
- $\forall i_1, i_2 \in T_{co}$ such that $i_1 \prec_{co} i_2$, we have $i_1 \otimes_{co} i_2 = i$, where $s(i) = s(i_1)$ and $e(i) = e(i_2)$. ■

The sequencing operator of CORE may be defined as follows:

Definition 12 (Sequencing Operator in CORE)**.** Consider two activities α_1 and α_2 and a stream S. We have:

$$[[\alpha_1\,;\alpha_2]]^{co}_S = \{i_1 \otimes_{co} i_2 \mid i_1 \in [[\alpha_1]]^{co}_S \wedge i_2 \in \mathsf{S}_{co}(i_1, [[\alpha_2]]^{co}_S)\} \quad ■$$

The sequencing operator of CORE cannot be used in RTEC because it violates Requirement 1. Consider an input stream S, where activity α_1 holds in $i_{11} = (1,3)$ and $i_{12} = (5,6)$, activity α_2 holds in $i_2 = (8,11)$, and activity α_3 holds in $i_3 = (15,22)$. Based on Definitions 11 and 12, $[[\alpha_1\,;\alpha_3]]^{co}_S$ contains intervals $(1,22)$ and $(5,22)$, thus not being a set of MDIs.

The language of CORE supports a collection of operators implementing so-called "selection strategies", which impose constraints on the detected composite activities [18]. We leave the study of sequencing with selection strategies for future work.

Wayeb is an another state-of-the-art automata-based CER engine. While there are similarities in the semantics of sequencing in Wayeb and CORE, the underlying formalisms of these frameworks differ in their temporal representation of composite activities. CORE marks a composite activity occurrence using an interval starting from the first item of the input stream that participates in the activity and ending at the last such item. In contrast, Wayeb represents a composite activity occurrence using the set of time-stamps of the input items of the stream that constitute the composite activity.

Definition 13 (Temporal Sequencing Model for Wayeb)**.** The temporal sequencing model for Wayeb is $(T_w, \prec_w, \mathsf{S}_w, \otimes_w)$, where

- T_w is a set containing all finite subsets of the positive integers.
- $\prec_w$ is a partial order on T_w, such that, for $\tau_1, \tau_2 \in T_w$, we have $\tau_1 \prec_w \tau_2$ iff $last(\tau_1) < first(\tau_2)$, i.e., the last time-point in τ_1 is less than the first time-point in τ_2.
- $\mathsf{S}_w(\tau_1, \mathrm{T}_2) = \{\tau_2 \in \mathrm{T}_2 \mid \tau_1 \prec_w \tau_2\}$.
- $\forall \tau_1, \tau_2 \in T_w$: $\tau_1 \prec_w \tau_2$, we have $\tau_1 \otimes_w \tau_2 = \tau_1 \cup \tau_2$. ■

Definition 14 (Sequencing Operator in Wayeb)**.** Consider two activities α_1 and α_2, and a stream S. We have:

$$[[\alpha_1\,;\alpha_2]]^w_S = \{\tau_1 \otimes_w \tau_2 \mid \tau_1 \in [[\alpha_1]]^w_S \wedge \tau_2 \in \mathsf{S}_w(\tau_1, [[\alpha_2]]^w_S)\} \quad ■$$

Wayeb represents activity occurrences with sets of time-points, and not intervals. Thus, it does not fulfill Requirement 1.

Suppose that, in an attempt to express an interval-based semantics for Wayeb, we were to revise its sequencing operator as: $rev[[\alpha_1\,;\alpha_2]]^w_S = \{(first(\tau), last(\tau)) \mid \tau \in [[\alpha_1\,;\alpha_2]]^w_S\}$. However, $rev[[\alpha_1\,;\alpha_2]]^w_S$ still fails at satisfying Requirement 1, because, for similar reasons as Definition 12, $rev[[\alpha_1\,;\alpha_2]]^w_S$ may not be composed of MDIs. Thus, $rev[[\alpha_1\,;\alpha_2]]^w_S$ is not suitable for RTEC.

7 RTEC$_\mathrm{S}$: RTEC with Sequencing

We propose RTEC$_\mathrm{S}$, i.e., an extension of RTEC that supports sequencing via the operator we introduced in Definition 8. We present the syntax, the semantics and the reasoning algorithms of RTEC$_\mathrm{S}$.

Syntax. Recall that activities are expressed in RTEC by means of fluent-value pairs (FVPs). RTEC$_\mathrm{S}$ extends RTEC with a sequencing construct in statically determined FVP definitions.

Definition 15 (Syntax of statically determined FVP definitions in RTEC$_\mathrm{S}$)**.** A $\mathsf{holdsFor}(F = V, I)$ rule defining a FVP $F = V$ may additionally contain body predicates in the form of $\mathsf{seq}(I_1, I_2, I)$, where I_1 and I_2 are input lists of MDIs, and I is an output list of MDIs. Given two FVPs $F_1 = V_1$ and $F_2 = V_2$ taking place in the MDIs of I_1 and I_2, I contains the MDIs during which the sequence of $F_1 = V_1$ and $F_1 = V_2$ takes place, following Definition 8. ■

Consider the following rule describing part of a fishing trip:

$$\begin{aligned}&\mathsf{holdsFor}(\mathit{fishingTripStart}(Vl) = \mathsf{true}, I) \leftarrow\\&\quad \mathsf{holdsFor}(\mathit{anchoredOrMoored}(Vl) = \mathsf{true}, I_{am}),\\&\quad \mathsf{holdsFor}(\mathit{withinArea}(Vl, \mathit{fishing}) = \mathsf{true}, I_f),\\&\quad \mathsf{seq}(I_{am}, I_f, I).\end{aligned} \tag{4}$$

FVP $\mathit{anchoredOrMoored}(Vl) = \mathsf{true}$ denotes that vessel Vl is either anchored or moored near some port (rule (3)), while FVP $\mathit{withinArea}(Vl, \mathit{fishing}) = \mathsf{true}$ expresses that Vl is within a fishing area (rules (1) and (2)). In rule (4), $\mathsf{seq}(I_{am}, I_f, I)$ computes the list of MDIs I where the vessel is said to be at first anchored or moored, and then within a fishing area, indicating the start of a fishing trip.

Semantics. The introduction of seq in holdsFor rules does not affect the definition of a dependency graph (Definition 2). Therefore, our extension of RTEC does not affect its semantics.

Proposition 4 (Semantics of RTEC$_\mathrm{S}$)**.** An event description in RTEC$_\mathrm{S}$ is a locally stratified logic program. ⧫

Reasoning. Algorithm 1 illustrates the steps followed by RTEC$_\mathrm{S}$ to compute $\mathsf{seq}(I_1, I_2, I)$. I in $\mathsf{holdsFor}(F = V, I)$ is a sorted list of MDIs (even if the items of the stream are not sorted) [4]. Therefore,

Algorithm 1 Sequencing in RTEC$_S$

Require: Sorted lists I_1 and I_2 of MDIs
Ensure: Sorted list I of MDIs

1: $j_1 \leftarrow 1, j_2 \leftarrow 1, I \leftarrow [\,]$
2: **while** $j_1 \leq |I_1|$ **and** $j_2 \leq |I_2|$ **do**
3: $\quad i_1 \leftarrow I_1[j_1], i_2 \leftarrow I[j_2]$
4: $\quad$ **if** $\mathsf{end}(i_2) < \mathsf{start}(i_1)$ **then** $j_2 \leftarrow j_2{+}1$
5: $\quad$ **else if** $j_1 = |I_1|$ **then** $\quad \triangleright$ we have $i_1 \prec_{rt} i_2$ hereafter
6: $\qquad I.\mathsf{append}(i_1 \otimes_{rt} i_2)$, **return** I
7: $\quad$ **else** $\quad \triangleright$ j_1 is not pointing to the last interval in I_1
8: $\qquad i_1^{next} \leftarrow I_1[j_1{+}1]$
9: $\qquad$ **if** $\mathsf{end}(i_2) < \mathsf{start}(i_1^{next})$ **then**
10: $\qquad\quad I.\mathsf{append}(i_1 \otimes_{rt} i_2), j_2 \leftarrow j_2{+}1$
11: $\qquad j_1 \leftarrow j_1{+}1$
12: **return** I

I_1 and I_2 in $\mathsf{seq}(I_1, I_2, I)$ are also sorted lists of MDIs (see Definition 1). In order to compute I, we iterate over interval pairs from the lists of MDIs I_1 and I_2, following an ascending temporal order, using indices j_1 and j_2 (see lines 1–3). Our goal is to find interval pairs that are adjacent based on Definition 6, so that we may construct the intervals in I by composing these adjacent intervals using $\otimes_{rt}$.

For an interval pair $i_1 \in I_1$ and $i_2 \in I_2$, if i_2 ends before the start of i_1, i.e., $i_2 \prec_{rt} i_1$, then i_1 and i_2 are not adjacent, and, since I_1 and I_2 are sorted in ascending temporal order, i_2 may not be adjacent with any interval in I_1 that is after i_1. Thus, we move to the next interval in I_2 (line 4). Otherwise, if $i_2 \not\prec_{rt} i_1$, then, since i_1 and i_2 are non-overlapping (see Assumption 2), we have that $i_1 \prec_{rt} i_2$. In this case, based on the interval ordering in I_1 and I_2, we are certain that i_2 is the earliest interval in I_2 that is after i_1, i.e., there is no interval $i_2' \in I_2$ such that $i_1 \prec_{rt} i_2' \prec_{rt} i_2$. Therefore, in order to check whether i_1 and i_2 are adjacent, it suffices to examine whether there is an interval $i_1' \in I_1$ such that $i_1 \prec_{rt} i_1' \prec_{rt} i_2$. There are two cases: If i_1 is the last interval in I_1, then there is no interval i_1' in I_1 such that $i_1 \prec_{rt} i_1'$, and thus we add $i_1 \otimes_{rt} i_2$ in I and return I (lines 5–6). Otherwise, if i_1 is not the last interval in I_1, we check whether the interval i_1^{next} that is right after i_1 in I_1 satisfies $i_2 \prec_{rt} i_1^{next}$. If so, then there is no interval in I_1 that is both after i_1 and before i_2. Therefore intervals i_1 and i_2 are adjacent; we add $i_1 \otimes_{rt} i_2$ in I and increment j_1 and j_2, towards identifying the next MDI for I, if any (lines 7–11). In the case that $i_2 \not\prec_{rt} i_1^{next}$, which implies that $i_1 \prec_{rt} i_1^{next} \prec_{rt} i_2$—and thus i_1 and i_2 are not adjacent—we increment j_1 and not j_2 (line 11). This is because i_2 may be adjacent with an interval of I_1 that is after i_1, and thus should be considered in the next iteration. We return I when all intervals in I_1 or I_2 have been processed (line 2). Afterwards, RTEC$_S$ caches list I in order to be able to resolve patterns requiring I very efficiently.

Example 6 (Sequencing in RTEC$_S$)**.** Consider two sorted lists of MDIs $I_1 = [i_{11}, i_{12}]$ and $I_2 = [i_{21}, i_{22}]$, where $i_{11} = (8, 9)$, $i_{12} = (12, 18)$, $i_{21} = (1, 3)$ and $i_{22} = (25, 26)$. We outline an execution of Algorithm 1 on I_1 and I_2, leading to an output list of MDIs I. We start by processing interval pair i_{11} and i_{21}, i.e., index j_1 points to i_{11} and index j_2 points to i_{21} (lines 1–3). Since i_{11} is after i_{21}, i_{11} is not adjacent with i_{21}, and we move j_2 over the next interval of I_2, i.e., i_{22} (line 4). The next interval pair consists of i_{11} and i_{22}; i_{11} is not adjacent with i_{22} because $i_{11} \prec_{rt} i_{12} \prec_{rt} i_{22}$. Thus, we move j_1 over the next interval of I_1, i.e., i_{12} (lines 7–9 and 11). Subsequently, we verify that i_{12} is adjacent with i_{22}, because i_{12} is before i_{22} and there is no interval in I_1 that is between i_{12} and i_{22}—in fact, i_{12} is the last interval in I_1—and thus we add interval $i_{12} \otimes_{rt} i_{22} = (12, 26)$ in list I, and return I (lines 5–6). $\diamondsuit$

Proposition 5 (Correctness of Sequencing in RTEC$_S$)**.** Consider activities α_1 and α_2, and a stream S. Given the sorted lists of MDIs I_1 and I_2 of α_1 and α_2, RTEC$_S$ computes a list of MDIs I for $\alpha_1 ; \alpha_2$ such that $i \in I$ iff $i \in [[\alpha_1 ; \alpha_2]]^{rt}_S$. $\blacklozenge$

Proposition 6 (Complexity of Sequencing in RTEC$_S$)**.** Consider activities α_1 and α_2, and a stream S. The worst-case time complexity of RTEC$_S$ for computing the MDIs of $\alpha_1 ; \alpha_2$ is $O(|[[\alpha_1]]^{rt}_S| + |[[\alpha_2]]^{rt}_S|)$. $\blacklozenge$

Proposition 6 states that RTEC$_S$ computes sequencing patterns with a single pass over the streaming data.

8 RTEC$_S$ with Windowing

To handle streaming applications, RTEC operates in a windowing mode, i.e., at each 'query time' q_j, it takes into consideration the items of the input stream S that fall within a specified sliding window with size ω [4]. All items of S that took place before or at $q_j - \omega$ are discarded/'forgotten'. Using windowing, reasoning efficiency depends on the size ω, instead of the size of S, leading to significant cost reductions. In this section, we outline the conditions under which RTEC$_S$ performs correct reasoning over windows.

A window $w = (q_j - \omega, q_j]$ delimits a finite, continuous subset S_w of a stream S on which temporal pattern matching may be performed. For an input activity α, the set of occurrences $[[\alpha]]^{rt}_{S_w}$ of α in S_w is composed of all intervals $i \cap w$ such that $i \in [[\alpha]]^{rt}_S$. Given activities α_1 and α_2, computing $\alpha_1 ; \alpha$ over S_w is correct iff $[[\alpha_1 ; \alpha_2]]^{rt}_{S_w}$ contains the intervals in set $\{i \cap w \mid i \in [[\alpha_1 ; \alpha_2]]^{rt}_S\}$, i.e., the set of intervals produced by evaluating $\alpha_1 ; \alpha_2$ over the entire stream S, and then keeping only the intersection of each of these intervals with w. We use $[[\alpha_1 ; \alpha_2]]^{rt}_S {\downarrow} w$ as a shorthand for this set.

Proposition 7 (Correctness of Sequencing over Windows)**.** Consider a window w over a stream S, and activities α_1 and α_2. Moreover, suppose that i_f and i_l are, respectively, the earliest and the most recent interval in $[[\alpha_1]]^{rt}_{S_w} \cup [[\alpha_2]]^{rt}_{S_w}$. $[[\alpha_1 ; \alpha_2]]^{rt}_{S_w} = [[\alpha_1 ; \alpha_2]]^{rt}_S {\downarrow} w$ if $i_f \in [[\alpha_1]]^{rt}_{S_w}$ and $i_l \in [[\alpha_2]]^{rt}_{S_w}$. $\blacklozenge$

If the earliest interval i_f of α_1 or α_2 in a window w is an interval of α_1, then there is a no interval i_1 of α_1 before w that is adjacent to an interval i_2 of α_2 in w, because $i_1 \prec_{rt} i_f \prec_{rt} i_2$. Thus, there is no interval of $\alpha_1 ; \alpha_2$ that overlaps the start of w and is not included $[[\alpha_1 ; \alpha_2]]^{rt}_{S_w}$. For similar reasons, we guarantee correctness when the latest interval of α_1 or α_2 in w is an interval of α_2.

In the case that i_f is an interval of α_2, then there may be an interval i_1 of α_1 before w that is adjacent with i_f, implying that $[i_1 \otimes_{rt} i_f] \cap w \in [[\alpha_1 ; \alpha_2]]^{rt}_S {\downarrow} w$ and $[i_1 \otimes_{rt} i_f] \cap w \notin [[\alpha_1 ; \alpha_2]]^{rt}_{S_w}$. We may avoid such false negatives by caching intervals of activities appearing in the left-hand side of sequencing operators.

Corollary 1 (Interval Caching for Sequencing)**.** Consider a stream S, a window w, and activities α_1 and α_2. If the most recent interval i_l in $[[\alpha_1]]^{rt}_{S_w} \cup [[\alpha_2]]^{rt}_{S_w}$ is an interval of α_1 and $i_l \otimes_{rt} i' \in [[\alpha_1 ; \alpha_2]]^{rt}_S$, then caching i_l in a memory C guarantees that that there is a window w' after w such that $i_l \otimes_{rt} i' \in [[\alpha_1 ; \alpha_2]]^{rt}_{S_{w'} \cup C}$. $\triangle$

9 Experimental Analysis

9.1 Experimental Setup

We present an experimental evaluation of RTEC$_S$, including a comparison with CORE, i.e., a state-of-the-art CER system with highly optimised pattern matching techniques [9]. We did not equip CORE

Parameters		Reasoning Time		Computed Intervals	
N	D	RTEC$_S$	CORE	RTEC$_S$	CORE
3	10K	19	1K	500	148K
6	10K	23	7K	402	669K
12	10K	30	2K	35	109K
3	50K	82	109K	500	19M
6	50K	87	>600K	500	>30M
12	50K	107	>600K	500	>30M

	Reasoning Time			Computed Intervals		
N	RTEC$_S$	RTEC$_{S\text{-}f}$	CORE	RTEC$_S$	RTEC$_{S\text{-}f}$	CORE
3	31	39	2K	1.5K	1.5K	193K
6	63	84	18K	6.6K	6.6K	1.5M
12	240	400	48K	12.4K	12.4K	1.6M

Window Size		Reasoning Time	Computed Intervals
Days	D	RTEC$_S$	RTEC$_S$
1	73K	2K	18K
2	145K	6K	33K
4	272K	14K	61K
8	545K	32K	119K
16	1M	79K	236K

Table 1. CER over one abstract sequencing pattern (left), multiple abstract sequencing patterns (middle) and real maritime activities (right). We evaluated only RTEC$_S$ on the maritime event description because there is no other CER engine that supports both sequencing and inertial activities. N and D denote the number of input activity types and the number of input activity instances in the dataset/window. Time is in milliseconds.

with any selection strategy. We constructed a domain with abstract activities, and generated synthetic datasets for this domain, for the purpose of stress testing RTEC$_S$ and CORE. The event description of this domain includes sequencing patterns on activities with the same Id, such as $\alpha_1(Id)\,;\alpha_2(Id)\,;\alpha_3(Id)$. Moreover, we employed an event description for maritime situational awareness, defining maritime activities using both simple and statically determined fluents. We used streams of events that were derived from Automatic Identification System (AIS) signals, containing information about vessels' location, speed and heading. The task was to compute intervals for various types of dangerous, suspicious or illegal vessel activities, such as a ship-to-ship transfer of goods in the open sea [31]. We used a publicly available dataset [https://zenodo.org/record/1167595], containing 18M AIS signals, emitted by 5K vessels sailing around the port of Brest, France, between October 2015–March 2016.

Our experiments are reproducible; the code of RTEC$_S$, as well as the datasets and the patterns we used, are publicly available[3]. RTEC$_S$ operated on SWI-8.4 Prolog, while CORE was evaluated using C++23 with the Clang++ compiler. Both engines ran on a PC with Ubuntu 22, Ryzen 7 5700U and 16GB RAM.

9.2 *Experimental Results*

In our first set of experiments, we evaluated RTEC$_S$ and CORE on datasets from our abstract activity domain, including 500 possible entity ids. The sequencing pattern was $\alpha_1(Id)\,;\alpha_2(Id)\,;\ldots;\alpha_N(Id)$. Table 1 (left) presents our results for values of N ranging from 3 to 12. First, we ran RTEC$_S$ and CORE over datasets of 10K activities. Both systems operated directly over the entire dataset, i.e., no windows were used. Our results show that RTEC$_S$ is faster than CORE by orders of magnitude. For each entity id, CORE computed all possible sequencing combinations on the input activities, leading to thousands of overlapping intervals, whereas RTEC$_S$ considered only combinations of adjacent activity intervals, which were drastically fewer. To stress test further RTEC$_S$ and CORE, we evaluated them on a dataset containing 50K abstract activities, again with 500 possible entity ids. The results are in the last 3 rows of Table 1 (left). Due to the increased number of input activities, the number of possible combinations of activities with the same id rose sharply, leading to an explosion in the number of intervals computed by CORE, as well as its reasoning time. In contrast, RTEC$_S$ focused only on adjacent intervals, leading to much more stable performance.

In our second set of experiments, our goal was to investigate the potential benefits of the caching mechanism of RTEC$_S$ when processing hierarchies of sequencing patterns. We employed event descriptions including abstract activities and multiple patterns. For each value of parameter N, we constructed an event description that was composed of pattern $\alpha_1(Id)\,;\alpha_2(Id)\,;\ldots;\alpha_N(Id)$, as well as all its possible subpatterns. In the case of $N=3$, e.g., the event description included patterns $\alpha_1(Id)\,;\alpha_2(Id)\,;\alpha_3(Id)$, $\alpha_1(Id)\,;\alpha_2(Id)$ and $\alpha_2(Id)\,;\alpha_3(Id)$. By construction, the patterns in these event descriptions share several common subpatterns, indicating that we may benefit from compositionality and caching intermediate results. To investigate such benefits, we included in our evaluation RTEC$_{S\text{-}f}$, i.e., a version of RTEC$_S$ that flattens the activity hierarchy before reasoning, and thus, contrary to RTEC$_S$, is unable to cache intermediate results. CORE also lacks such a caching mechanism. We evaluated RTEC$_S$, RTEC$_{S\text{-}f}$ and CORE on such event descriptions, where N ranged from 3 to 12. The datasets included 10K input activities and 500 entity ids. Table 1 (middle) displays our results. We observe that RTEC$_S$ was more efficient than RTEC$_{S\text{-}f}$ at MDI computation. For $N=12$, i.e., our largest activity hierarchy, including 66 patterns, RTEC$_S$ yielded a 40% benefit in reasoning efficiency compared to RTEC$_{S\text{-}f}$, while computing the same MDIs as RTEC$_{S\text{-}f}$. Similarly to the previous experiments, CORE computed large numbers of overlapping intervals, requiring much more reasoning time than RTEC$_S$.

The goal of our final set of experiments was to test the efficacy of RTEC$_S$ on a real domain, including millions of input activities and an event description with both inertial and sequential activity definitions. To do this, we evaluated RTEC$_S$ on real maritime data from the Brest area. We could not include CORE in these experiments because it does not support inertial activities. We evaluated RTEC$_S$ for an increasing window size, spanning from 1 to 16 days. Table 1 (right) displays our results, demonstrating that RTEC$_S$ is able to detect both inertial and sequential activities over large, real data streams efficiently. For the largest window of 16 days we employed, which included, on average, about 1 million input activities, RTEC$_S$ was able to compute about 236K MDIs for composite maritime activities per window, requiring an average reasoning time of about 79 seconds.

10 Summary and Further Work

We presented RTEC$_S$, an extension of the CER engine RTEC with a sequencing operator. RTEC$_S$ is the only CER system that captures both inertial and sequential phenomena. We assessed our sequencing operator theoretically, proving compositionality, associativity and correctness, and demonstrating its benefits for interval-based CER with activity hierarchies. Moreover, we presented a reproducible empirical evaluation of RTEC$_S$ on artificial and real data, including a comparison with CORE, showcasing the caching and windowing features of RTEC$_S$, which are essential for CER.

In the future, we will implement the caching mechanism that guarantees correctness of sequencing over windows.

[3] https://github.com/Periklismant/rtecs_ecai25_supplementary

Acknowledgements

Periklis Mantenoglou was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Alexander Artikis was supported by the EU-funded CREXDATA project (No 101092749).

References

[1] E. Alevizos, A. Artikis, and G. Paliouras. Complex event recognition with symbolic register transducers. *Proc. VLDB Endow.*, 17(11):3165–3177, 2024.

[2] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu. ATLAS: A sequence-based learning approach for attack investigation. In M. D. Bailey and R. Greenstadt, editors, *USENIX*, pages 3005–3022, 2021.

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

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

[5] A. Artikis, N. Katzouris, I. Correia, C. Baber, N. Morar, I. Skarbovsky, F. Fournier, and G. Paliouras. A prototype for credit card fraud management: Industry paper. In *DEBS*, pages 249–260. ACM, 2017.

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

[7] P. Baumgartner. Combining event calculus and description logic reasoning via logic programming. In *FroCoS*, pages 98–117, 2021.

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

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

[10] F. Chesani, P. Mello, M. Montali, and P. Torroni. Representing and monitoring social commitments using the event calculus. *Auton. Agents Multi Agent Syst.*, 27(1):85–130, 2013.

[11] L. Chittaro and A. Montanari. Efficient temporal reasoning in the cached event calculus. *Comput. Intell.*, 12(3):359–382, 1996.

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

[13] A. J. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. M. White. Towards expressive publish/subscribe systems. In *EDBT*, volume 3896, pages 627–644. Springer, 2006.

[14] C. Dousson and P. L. Maigat. Chronicle recognition improvement using temporal focusing and hierarchisation. In *IJCAI*, pages 324–329, 2007.

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

[16] A. Galton and J. C. Augusto. Two approaches to event definition. In *DEXA*, volume 2453, pages 547–556, 2002.

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

[18] A. Grez, C. Riveros, M. Ugarte, and S. Vansummeren. A formal framework for complex event recognition. *ACM Trans. Database Syst.*, 46 (4):16:1–16:49, 2021.

[19] Ö. Kafali, A. E. Romero, and K. Stathis. Agent-oriented activity recognition in the event calculus: An application for diabetic patients. *Comput. Intell.*, 33(4):899–925, 2017.

[20] I. Kolchinsky and A. Schuster. Real-time multi-pattern detection over event streams. In *SIGMOD*, pages 589–606. ACM, 2019.

[21] M. Körber, N. Glombiewski, A. Morgen, and B. Seeger. TPStream: low-latency and high-throughput temporal pattern matching on event streams. *Distributed Parallel Databases*, 39(2):361–412, 2021.

[22] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Computing*, 4(1):67–96, 1986.

[23] S. Liu, H. Dai, S. Song, M. Li, J. Dai, R. Gu, and G. Chen. ACER: Accelerating complex event recognition via two-phase filtering under range bitmap-based indexes. In *KDD*, pages 1933–1943. ACM, 2024.

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

[25] P. Mantenoglou, D. Kelesis, and A. Artikis. Complex event recognition with allen relations. In *KR*, pages 502–511, 2023.

[26] M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. P. van der Aalst. Monitoring business constraints with the event calculus. *ACM Trans. Intell. Syst. Technol.*, 5(1):17:1–17:30, 2013.

[27] A. Paschke. ECA-RuleML: An approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics. Technical Report 11, TU München, 2005.

[28] A. Paschke and M. Bichler. Knowledge representation concepts for automated SLA management. *Decis. Support Syst.*, 46(1):187–205, 2008.

[29] A. Paschke and M. Bichler. Knowledge representation concepts for automated SLA management. *Decision Support Systems*, 46(1):187–205, 2008.

[30] D. Pinto and C. Riveros. Complex event recognition meets hierarchical conjunctive queries. *Proc. ACM Manag. Data*, 2(5):216:1–216:26, 2024.

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

[32] T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1988.

[33] P. A. Walega, M. Kaminski, D. Wang, and B. C. Grau. Stream reasoning with DatalogMTL. *J. Web Semant.*, 76:100776, 2023.

[34] D. Wang, P. Hu, P. A. Walega, and B. C. Grau. MeTeoR: Practical reasoning in datalog with metric temporal operators. In *AAAI*, pages 5906–5913, 2022.

[35] W. M. White, M. Riedewald, J. Gehrke, and A. J. Demers. What is "next" in event processing? In *PODS*, pages 263–272. ACM, 2007.