Stream Reasoning with Sequencing

Manolis Pitsikalis[1] and Alexander Artikis[2,1]

[1] NCSR Demokritos, Athens, Greece, manospits@iit.demokritos.gr
[2] University of Piraeus, Greece, a.artikis@unipi.gr

Abstract

We present $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$, an interval-based stream reasoning system. $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ inherits from its predecessor RTEC_s a compositional and associative sequencing operator. $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ extends RTEC_s with 'Reasoning Cut-off Tuples' (RCTs), a mechanism allowing for monotonic output given a monotonic input. $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ is also equipped with a bookkeeping policy, guided by RCTs, that identifies the minimal information for correct reasoning. Our empirical, reproducible evaluation on real and synthetic data from three application domains demonstrates the efficacy of $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$.

1 Introduction

Stream reasoning systems continuously apply temporal patterns/queries on incoming data and report instances of pattern satisfaction [11]. 'Complex Event Recognition' (CER) systems, for example, reason over high-velocity streams in order to detect instances of (spatio-)temporal pattern satisfaction with minimal latency [20]. In the maritime domain, CER systems consume streams of vessel position signals, in order to detect instances of dangerous, suspicious and illegal vessel activities at run-time, thus supporting safe shipping. The most common operator in CER is 'sequencing', expressing phenomena where activities take place one after the other [14]. In the maritime domain, a potentially illegal transhipment may be identified through the sequence of the following activities: first, a vessel is engaged in fishing, such as trawling; then, a 'rendezvous' takes place, i.e. the vessel is in proximity with another vessel in the open sea for a relatively long period of time, which may indicate a transfer of the catch. Several frameworks express sequencing only over instantaneous activities, and fail to support durative phenomena, which are common in CER [6, 23]. 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, i.e. when the pattern expressing a complex event depends on other complex event patterns [19, 32, 48].

Most CER systems operate on 'monotonic input' — they assume that incoming events are temporally ordered. Unfortunately, CER systems provide no formal guarantee on the stability of their output; they do not identify the largest prefix of the output that will remain invariant under any possible evolution of the input stream. In the presence of sequencing, this omission is critical as pattern satisfaction may depend on events that have not yet arrived. As a result, previously reported derivations may require retrospective revision, undermining the reliability of the reasoning process. This issue becomes more pronounced in the common case of hierarchical patterns because errors in earlier derivations, due to information that was not available, and retrospective revisions, may propagate through the hierarchy, increasing reasoning cost and compromising decision-making.

We present a stream reasoning system that addresses these issues. Our starting point is RTEC_s [28], an extension of the Run-Time Event Calculus (RTEC) [5] with support for compositional and associative interval-based sequencing. As an implementation of the Event Calculus [24], the language of RTEC_s has built-in support for inertial phenomena. A formalisation

in $\mathsf{RTEC_s}$ is a (locally) stratified logic program [35], and includes relational rules that may refer to background knowledge. Moreover, $\mathsf{RTEC_s}$ inherits from RTEC a set of interval manipulation constructs and reasoning techniques, such as indexing and incremental caching, that allow it to scale to high-velocity streams.

Given a monotonic input, the output of RTEC is monotonic, i.e. the computed intervals of the phenomena of interest will never be revised in the future. The introduction of sequencing, however, does not guarantee monotonic output. To address this issue, we present $\mathsf{RTEC_s^{RCT}}$, an extension of $\mathsf{RTEC_s}$ with monotonic output over a monotonic stream. $\mathsf{RTEC_s^{RCT}}$ employs 'Reasoning Cut-off Tuples' (RCTs), a construct that may be used to determine the latest time-point up to which the output of reasoning remains unaffected by the arrival of future data. RCT computation is not restricted to patterns defined in terms of sequence; $\mathsf{RTEC_s^{RCT}}$ has mechanisms for RCT computation for all interval manipulation constructs. This way, $\mathsf{RTEC_s^{RCT}}$ supports monotonic output for arbitrary pattern hierarchies.

Stream reasoning systems typically employ windowing techniques in order to restrict reasoning complexity. RCTs allow $\mathsf{RTEC_s^{RCT}}$ to perform stateful windowing with respect to sequencing, i.e. RCTs allow us to identify the minimal historical information that should be cached for correct and efficient reasoning in the future.

The contributions of the paper are then the following. First, we extend $\mathsf{RTEC_s}$ with RCTs — given a monotonic stream, these allow us to identify the largest prefix of the output that will not change in the future. This way, the resulting system, $\mathsf{RTEC_s^{RCT}}$, avoids the uncertainty stemming from non-monotonic output. Second, we propose a bookkeeping policy, guided by RCTs, for stateful, correct and efficient windowing. Third, we present an empirical evaluation on maritime situational awareness, commercial fleet management and multi-agent system monitoring, using real and synthetic datasets — the instructions to reproduce the experiments are available with the code of $\mathsf{RTEC_s^{RCT}}$: `https://github.com/aartikis/RTEC/tree/sRCT`. Our empirical analysis shows that $\mathsf{RTEC_s^{RCT}}$ outperforms significantly $\mathsf{RTEC_s}$, and may support runtime decision-making in large-scale real applications. The proofs of the propositions presented in the paper may be found in the supplementary material in the repository of $\mathsf{RTEC_s^{RCT}}$.

2 Background: $\mathsf{RTEC_s}$

We present an overview of the language, semantics and reasoning techniques of $\mathsf{RTEC_s}$ [28]. To aid understanding, we employ a running example from Maritime Situational Awareness (MSA).

2.1 Language

Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. The time model $\mathcal{T}$ is linear and includes the natural numbers, i.e. $\mathcal{T}=\langle\mathbb{N},>\rangle$. The set of intervals over $\mathcal{T}$ is $\mathcal{I}=\{[t_s,t_e)|t_s,t_e{\in}\mathbb{N}\wedge t_s{<}t_e\}$. The specification of a domain in $\mathsf{RTEC_s}$ includes domain objects, 'events', i.e. phenomena that may change the state of the world, and 'fluents', which are domain properties that may have different values at different points in time.

Definition 1 (Domain in $\mathsf{RTEC_s}$)**.** *A domain in* $\mathsf{RTEC_s}$ *is a tuple* $D=(\mathcal{O},\mathcal{E},\mathcal{F},\mathcal{V})$*, where* $\mathcal{O}$ *is a set of constants expressing the objects of the domain,* $\mathcal{E}$ *and* $\mathcal{F}$ *are sets of functor symbols describing the event and fluent types of the domain, and* $\mathcal{V}$ *is a set containing a set* $\mathcal{V}^F$ *for each* $F\in\mathcal{F}$*, comprising constants that denote the possible values of fluent* F*. All sets in* D *are finite.*

Definition 2 (Event). *An event E of domain $D=(\mathcal{O},\mathcal{E},\mathcal{F},\mathcal{V})$ is a term with functor symbol $e\in\mathcal{E}$, and arguments a_1, a_2, …, a_n, which may be variables or elements of $\mathcal{O}$, where n is the arity of e.*

Definition 3 (Fluent-Value Pair (FVP)). *A fluent F of domain $D=(\mathcal{O},\mathcal{E},\mathcal{F},\mathcal{V})$ is a term with functor symbol $f\in\mathcal{F}$, and arguments a_1, a_2, …, a_n, which may be variables or elements of $\mathcal{O}$, where n is the arity of f. A value V for fluent F is an element of set $\mathcal{V}^F$. $F{=}V$ is a fluent-value pair (FVP) expressing that fluent F has value V.*

Example 1 (Events and FVPs in MSA). *An* $\mathsf{RTEC_s}$ *domain for MSA is $D=(\mathcal{O}_m,\mathcal{E}_m,\mathcal{F}_m,\mathcal{V}_m)$, where $\mathcal{O}_m$ contains constants denoting e.g. vessel identifiers and area types, $\mathcal{E}_m$ contains the types of events, such as 'entersArea', $\mathcal{F}_m$ contains fluent types expressing vessel activities, such as 'stopped', and $\mathcal{V}_m$ contains the set of possible values for each fluent type. For instance, $\mathcal{V}_m^{stopped}$ contains 'nearPorts' and 'farFromPorts'. The events and FVPs of MSA are constructed by instantiating the event and fluent types in $\mathcal{E}_m$ and $\mathcal{F}_m$ with objects from $\mathcal{O}_m$ or variables. For example, the FVP 'stopped$(v_{17})=$farFromPorts' expresses that vessel v_{17} has stopped in the open sea.*

$\mathsf{RTEC_s}$ expresses event occurrences and FVPs over time using a set of domain-independent predicates.

Definition 4 ($\mathsf{RTEC_s}$ Predicates). $\mathsf{RTEC_s}$ *employs the following predicates:*

- $\mathsf{happensAt}(E,T)$*: event E occurs at time-point T.*
- $\mathsf{initiatedAt}(F=V,T)$*: a time period during which fluent F has value V is initiated at time-point T.*
- $\mathsf{terminatedAt}(F=V,T)$*: a time period during which fluent F has value V is terminated at time-point T.*
- $\mathsf{holdsAt}(F=V,T)$*: fluent F has value V at time-point T.*
- $\mathsf{holdsFor}(F=V,I)$*: fluent F has value V continuously during the maximal intervals in list I.*
- $\mathsf{iConstruct}(L_i,I)$ *i.e. interval manipulation construct* $\mathsf{union\_all}(L_i,I)$, $\mathsf{intersect\_all}(L_i,I)$, *or* $\mathsf{relative\_complement\_all}(L_i[0],L_i[1:],I)$*. L_i is the list of input lists of maximal intervals, and I is the output list of maximal intervals. See Figure 1-left for an illustration.*
- $\mathsf{seq}(I_a,I_b,I)$*: I is the list of maximal intervals expressing the sequence between the intervals of lists I_a and I_b.*

To present the semantics of the sequence operator seq, we first need to define the conditions in which two intervals are adjacent. In what follows, $s(i)$ and $e(i)$ denote the starting point and the ending point of interval i, and $i_1 \prec i_2$ iff $e(i_1)<s(i_2)$.

Definition 5 (Adjacent intervals). *The adjacent interval of interval $i_1\in I_1$ from interval list I_2 is given by*

$$\mathbb{A}(i_1,I_1,I_2)=\begin{cases}\{i_2\}, & \text{if } \exists i_2\in I_2.\ i_1\prec i_2\\ & \wedge\ (\neg\exists i_2'\in I_2.\ i_1\prec i_2'\prec i_2)\\ & \wedge\ (\neg\exists i_1'\in I_1.\ i_1\prec i_1'\prec i_2)\\ \emptyset, & \text{otherwise.}\end{cases}$$

Definition 6 (Sequence). *Given two disjoint maximal interval lists I_1 and I_2, I in* $\mathsf{seq}(I_1,I_2,I)$ *is defined as*

$$I=\{i_1\otimes i_2 \mid i_1\in I_1\wedge i_2\in I_2\wedge \mathbb{A}(i_1,I_1,I_2)=\{i_2\}\}$$

where $i_1\otimes i_2=i$ s.t. $s(i)=s(i_1)$ and $e(i)=e(i_2)$.

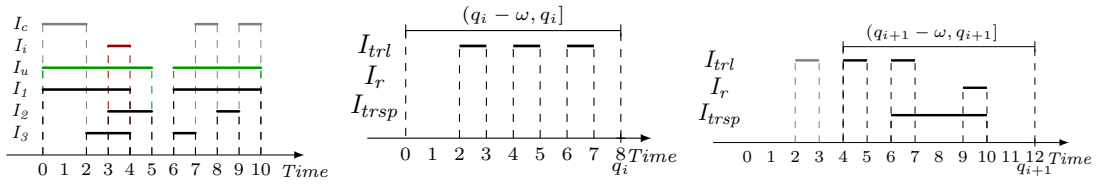


Figure 1: Left: Interval manipulation constructs; I_1, I_2 and I_3 are input lists of maximal intervals while I_c, I_i and I_u are the output lists of intervals for relative_complement_all$(I_1, [I_2, I_3], I_c)$, intersect_all$([I_1, I_2, I_3], I_i)$ and union_all$([I_1, I_2, I_3], I_u)$. Middle and right: illustration of *transhipment*; the gray line denotes a 'forgotten' interval.

The seq operator is both *associative* and *compositional* [28].

A formalisation in $\mathsf{RTEC_s}$ contains a set of domain-specific rules, called an *event description*.

Definition 7 (Event description)**.** *An event description $\mathcal{ED}$ is a tuple $\langle \mathcal{S}, \mathcal{H} \rangle$ where $\mathcal{S}$ is a set of rules with head* initiatedAt$(F = V, T)$ *or* terminatedAt$(F = V, T)$*, expressing the effects of events on FVPs; and $\mathcal{H}$ is a set of rules with head* holdsFor$(F = V, I)$ *defining $F{=}V$ in terms of other FVPs.*

Given a *stream* $\mathcal{N}$, i.e. a set of ground facts of the form happensAt(E, T), $\mathsf{RTEC_s}$ computes the maximal intervals during which FVPs hold. FVPs may be 'simple' or 'statically determined'. Simple FVPs are defined in terms of initiatedAt and terminatedAt rules, and are subject to the law of inertia, while statically determined FVPs are defined in terms of holdsFor rules. We restrict our attention to statically determined FVPs because the seq operator may only appear in the rules of such FVPs. The syntax of simple FVPs may be found in the repository of $\mathsf{RTEC_s^{RCT}}$.

Definition 8 (Statically determined FVP)**.** *A statically determined FVP $F{=}V$ is defined using a* holdsFor *rule with the following syntax:*

$$\begin{aligned} &\mathsf{holdsFor}(F = V, I_{n+m}) \leftarrow \\ &\quad \mathsf{holdsFor}(F_1 = V_1, I_1)[, \mathsf{holdsFor}(F_2 = V_2, I_2), \ \ldots, \ \mathsf{holdsFor}(F_n = V_n, I_n), \\ &\quad \mathsf{seq}(I_i, I_j, I_{n+1}) \mid \mathsf{iConstruct}(L_1, I_{n+1}), \ \ldots, \ \mathsf{seq}(I'_i, I'_j, I_{n+m}) \mid \mathsf{iConstruct}(L_m, I_{n+m})]]. \end{aligned} \tag{1}$$

We use the standard notation for logic programming [27], where e.g. ',' denotes conjunction and '$\leftarrow$' denotes implication. The first body literal of a holdsFor *rule defining $F = V$ is a* holdsFor *predicate expressing the maximal intervals of an FVP other than $F{=}V$. This is followed by a possibly empty list, denoted by '[[ ]]', of* holdsFor *predicates for other FVPs, interval manipulation constructs, expressed by* iConstruct *in formulation (1), or a sequence operator* seq.

Example 2 (Transhipment)**.** *Transhipment refers to the transfer of cargo between two vessels, often to extend operations without returning to the port. In the fishing industry, vessels may transfer their catch to refrigerated transport ships, allowing them to remain in fishing areas for longer periods. Although common, this practice complicates the tracking of fish origins and is sometimes associated with illegal activities. Consider the example definition below:*

$$\begin{aligned} &\mathsf{holdsFor}(\mathit{transhipment}(\mathit{Vl}_a, \mathit{Vl}_b) = \mathsf{true}, I_{trsp}) \leftarrow \\ &\quad \mathsf{holdsFor}(\mathit{trawling}(\mathit{Vl}_a) = \mathsf{true}, I_{trl}), \ \mathsf{holdsFor}(\mathit{rendezVous}(\mathit{Vl}_a, \mathit{Vl}_b) = \mathsf{true}, I_r), \\ &\quad \mathsf{seq}(I_{trl}, I_r, I_{trsp}). \end{aligned} \tag{2}$$

'transhipment(Vl_a, Vl_b)' is a Boolean statically determined fluent defined in terms of the 'trawling(Vl_a)' and 'rendezVous(Vl_a, Vl_b)' fluents. 'trawling(Vl_a)=true' holds in the intervals during which vessel Vl_a is fishing with the use of a fishing net, while 'rendezVous(Vl_a, Vl_b)=true' holds in the intervals during which vessels Vl_a and Vl_b are in proximity in the open sea for a long period of time. Thresholds such as those used to to determine whether two vessels remain in proximity for a sufficiently long period, are application-specific and are tuned by domain experts. Figure 1-right illustrates the computation of the 'transhipment' intervals. Sequencing requires the intervals of trawling and rendezVous to be adjacent. Thus only the intervals $[6, 7) \in I_{trl}$ and $[9, 10) \in I_r$ participate in a sequence.

Example 3 (Suspicious, anchored vessels)**.** *The rule below expresses a suspicious activity type.*

$$\begin{aligned}&\mathsf{holdsFor}(\mathit{suspiciousAnchored}(Vl)=\mathsf{true}, I) \leftarrow\\ &\quad \mathsf{holdsFor}(\mathit{transhipment}(\_, Vl)=\mathsf{true}, I_s),\ \mathsf{holdsFor}(\mathit{stopped}(Vl)=\mathit{farFromPorts}, I_{sf}),\\ &\quad \mathsf{holdsFor}(\mathit{withinArea}(Vl, \mathit{anchorage})=\mathsf{true}, I_a),\\ &\quad \mathsf{seq}(I_s, I_a, I_{sa}),\ \mathsf{intersect\_all}([I_a, I_{sa}, I_{sf}], I).\end{aligned} \tag{3}$$

'suspiciousAnchored(Vl)' is a Boolean fluent, 'stopped(Vl)' is a multi-valued fluent expressing the intervals during which vessel Vl is idle and takes the value 'nearPorts' or 'farFromPorts', depending on the vessel's distance from ports, 'withinArea(Vl, AreaType)' is a fluent expressing the intervals during which Vl is within an area of 'AreaType', and '_' denotes an unnamed variable. Rule (3) derives the intervals during which a vessel Vl is stopped within an anchorage area after being involved in transhipment, by applying first seq *on the lists of maximal intervals I_s and I_a, to produce I_{sa}, and then, by applying the* intersect_all *operation on the lists of maximal intervals I_a, I_{sa} and I_{sf}.*

2.2 Semantics and Run-Time Reasoning

An event description $\mathcal{ED}$ defines a *dependency graph* expressing the hierarchical relationships between the FVPs of $\mathcal{ED}$.

Definition 9 (Dependency graph)**.** *The dependency graph of an event description in* RTEC$_s$ *is a directed graph $G=(\mathcal{V}, \mathcal{E})$, where:*

- *$\mathcal{V}$ contains one vertex $v_{F=V}$ for each FVP $F=V$.*
- *$\mathcal{E}$ contains an edge $(v_{F=V}, v_{F'=V'})$ iff:*
  - *there is a rule with* initiatedAt$(F'=V', T)$ *or* terminatedAt$(F'=V', T)$ *as its head having* holdsAt$(F=V, T)$ *as one of its conditions, or*
  - *there is a* holdsFor *rule for $F'=V'$ having* holdsFor$(F=V, T)$ *as a condition.*

We may define a function *level* that maps all FVPs $F=V$ to the positive integers, as follows.

Definition 10 (Vertex Level)**.** *Given a directed acyclic graph, the level of vertex v is equal to:*

1. *1, if v has no incoming edges.*
2. *n, where $n>1$, if v has at least one incoming edge from a vertex of level $n-1$, while all its other incoming edges, if any, start from vertices with level $n-1$ or lower.*

In the absence of cycles in the dependency graph, the level of an FVP is equal to the level of its vertex. If a dependency graph contains cycles, the level of each FVP is first derived by contracting the vertices of the FVPs participating in each cycle into a single vertex, and then applying Definition 10 on the resulting contracted dependency graph. See [29] for the details.

The definition of FVP level allows us to arrange the ground atoms of an RTEC$_s$ event description into strata in ascending FVP level order. For FVPs with the same level, we introduce an additional stratum for each time-point, in ascending temporal order.

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

A proof may be constructed using the cycle-sum test [31], which establishes local stratification for logic programs with time arguments based on the temporal offsets between the head and the body conditions of each rule. Proposition 1 essentially indicates that $\mathsf{RTEC_s}$ inherits the semantics of locally stratified logic programs [35].

The key reasoning task of $\mathsf{RTEC_s}$ is the computation of $\mathsf{holdsFor}(F = V, I)$, i.e. the list of maximal intervals I during which an FVP $F{=}V$ of an event description holds continuously. Then, we may evaluate $\mathsf{holdsAt}$ as follows: for any fluent F, $\mathsf{holdsAt}(F = V, T)$ iff T belongs to one of the maximal intervals of I for which $\mathsf{holdsFor}(F = V, I)$. $\mathsf{RTEC_s}$ processes FVPs in a bottom-up manner, computing and caching their intervals level-by-level (Definition 10). This way, the intervals of the FVPs required for the processing of an FVP of level m are fetched from the cache without the need for re-computation. Furthermore, $\mathsf{RTEC_s}$ performs continuous query processing to compute the maximal intervals of FVPs. Given an event description $\mathcal{ED}$ and a stream $\mathcal{N}$, $\mathsf{RTEC_s}$ computes, at query-times q_i distanced by $step \in \mathbb{N}_{>0}$, the maximal intervals during which FVPs hold, taking into account the subset of the stream that overlaps a sliding temporal window of size $\omega \in \mathbb{N}_{>0}$. Windows in $\mathsf{RTEC_s}$ are *global*, i.e. all FVPs share the same window of the stream. At query-time q_i, $\mathsf{RTEC_s}$ will consider only events and FVP intervals in $(q_i{-}\omega, q_i]$, and 'forget' all earlier events and intervals (see Figure 1-middle and right). $\mathsf{RTEC_s}$ inherits from RTEC stateful windowing for simple fluents and statically determined fluents defined by means of interval manipulation constructs [4]. On the other hand, windowing for sequential phenomena is stateless, i.e. derivations from previous query-times are not reused [28].

Systems with a sequence operator in their language typically reason over monotonic streams, i.e. when the subset of the stream in $(q_i{-}\omega, q_i]$ is not revised after q_i. When necessary such systems sort the items of a subset of the input stream before processing them [20]. Given such a monotonic input, the output of RTEC is monotonic, i.e. the set of FVP intervals computed at query-time q_i will never be revised in the future. The introduction of sequencing, however, does not guarantee monotonic output. Consider Example 2 and Figure 1-middle and right. At q_i there is no interval for *rendezVous* so that the sequence between *trawling* and *rendezVous* can be satisfied. At q_{i+1} the sequence is satisfied and the interval $[6, 10)$ for *transhipment* is computed, with $6{<}q_i$. This non-monotonic behaviour can have significant consequences in (critical) applications. Following the previous example, in addition to false negatives at q_i, we may have false positives in FVPs further up in the hierarchy defined in terms of *transhipment* and $\mathsf{relative\_complement\_all}$. Such issues compromise the allocation of operational resources, undermining the reliability of run-time decision-support systems.

3 $\mathsf{RTEC_s^{RCT}}$

We present $\mathsf{RTEC_s^{RCT}}$, an extension of $\mathsf{RTEC_s}$ producing monotonic output given a monotonic input. Below, we use the term 'stream extension' or 'interval list extension' to refer to the addition of new intervals in the stream/interval list in the future.

3.1 Language and Semantics

$\mathsf{RTEC_s^{RCT}}$ operates on 'reasoning cut-off tuples' (RCTs) rather than lists of maximal intervals.

Definition 11 (Reasoning Cut-off Tuple). *An RCT R_o is a tuple $\langle I_o, T_o\rangle$, where I_o is a list of maximal intervals, and T_o is a positive integer such that $\forall i \in I_o$ we have $T_o \geq e(i)$. We call T_o of an RCT $R_o = \langle I_o, T_o\rangle$ the 'cut-off point' of R_o.*

The cut-off point may be viewed as a 'stability horizon': a guarantee that the intervals in I_o will not be revised by any future stream extension.

Example 4 (RCT for transhipment). *Recall Example 2. At query-time q_i (Figure 1-middle), we can be certain that the intervals $[2, 3)$ and $[4, 5)$ of I_{trl} will not participate in a sequence with an interval from I_r, as it is not possible to extend I_r in the future with an interval which is adjacent to these intervals. In contrast, an extension of the stream (as in Figure 1-right) might introduce an $i_r \in I_r$ adjacent to $[6, 7)$. Consequently, we can guarantee a monotonic output for transhipment at q_i only up to time-point 5, i.e. $R_{trsp} = \langle [\,], 5\rangle$ — '$[\,]$' denotes an empty list of intervals, indicating that there is no instance of transhipment up to time-point 5.*

An $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ event description is produced by translating every occurrence of interval manipulation operation, sequence and holdsFor of an RTEC_s event description into the corresponding construct operating on, and computing RCTs. We use the superscript $\bullet$ to denote the RCT-based version of a predicate, i.e. we use $\mathsf{iConstruct}^\bullet$, $\mathsf{seq}^\bullet$ and $\mathsf{holdsFor}^\bullet$ instead of iConstruct, seq and holdsFor. Consider e.g. the definition of *suspiciousAnchored* transformed for $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$:

$$\begin{aligned}&\mathsf{holdsFor}^\bullet(\mathit{suspiciousAnchored}(Vl) = \mathsf{true}, R) \leftarrow\\ &\quad \mathsf{holdsFor}^\bullet(\mathit{transhipment}(\_, Vl) = \mathsf{true}, R_s),\ \mathsf{holdsFor}^\bullet(\mathit{stopped}(Vl) = \mathit{farFromPorts}, R_{sf}),\\ &\quad \mathsf{holdsFor}^\bullet(\mathit{withinArea}(Vl, \mathit{anchorage}) = \mathsf{true}, R_a),\\ &\quad \mathsf{seq}^\bullet(R_s, R_a, R_{sa}),\ \mathsf{intersect\_all}^\bullet([R_a, R_{sa}, R_{sf}], R).\end{aligned} \tag{3'}$$

The second argument of $\mathsf{holdsFor}^\bullet$ and all arguments of $\mathsf{seq}^\bullet$ are RCTs. Moreover, the arguments of $\mathsf{iConstruct}^\bullet$, such as $\mathsf{intersect\_all}^\bullet$, are (lists of) RCTs.

The introduction of RCTs does not affect the definition of dependency graph (Definition 9). Thus, similar to RTEC_s event decriptions, we may arrange the ground atoms of an $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ event description into strata in ascending FVP level order, while for FVPs with the same level we may introduce an additional stratum for each time-point in ascending temporal order.

Proposition 2 (Semantics of $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$). *An event description in $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ is a locally stratified logic program.*

3.2 Reasoning with RCTs

RCTs allow us to determine, at each query-time, the largest prefix of the output that will not be modified under any extension of the input stream. In this section, we present the process of RCT computation, and in Section 3.3 we show how RCT computation is performed under windowing. Algorithm 1 illustrates RCT computation for $\mathsf{seq}^\bullet$ and $\mathsf{iConstruct}^\bullet$; these are denoted as $op^\bullet$ in Algorithm 1. Let $LR = [R_1, \ldots, R_n]$, with $R_i = \langle I_i, T_i\rangle$ and $i \in [1, n]$, be a list containing the input RCTs of the operation under consideration, and $R_\mathsf{o} = \langle I_\mathsf{o}, T_\mathsf{o}\rangle$ the output RCT. In the seq operation of rule (3') e.g. LR corresponds to $[R_s, R_a]$ and R_o corresponds to R_{sa}. The computation of the output RCT R_o requires three steps. First, we compute a candidate cut-off point T_o^c (line 1 of Algorithm 1). T_o^c is a time-point up to which the computed intervals for the operation in question will remain unchanged, regardless of any extension of the stream. However, there may be later cut-off points. Therefore, given T_o^c and the input RCTs, `extendedCTs` computes a possibly empty set E of cut-off points greater than T_o^c (line 2). The output cut-off point T_o is set to the latest cut-off point in $E \cup \{T_\mathsf{o}^c\}$ (line

Algorithm 1 `computeRCT`$(op^{\bullet}, InputRCTs)$

1: $T_o^c \leftarrow$ `computeCandidateCT`$(op^{\bullet}, InputRCTs)$
2: $E \leftarrow$ `extendedCTs`$(T_o^c, op^{\bullet}, InputRCTs)$
3: $T_o \leftarrow \max(E \cup \{T_o^c\})$
4: $TruncatedInpLs \leftarrow$ `cutInput`$(T_o, InputRCTs)$
5: $I_o \leftarrow \mathsf{RTEC_s}(op^{\bullet}, TruncatedInpLs)$
6: **return** $\langle I_o, T_o \rangle$

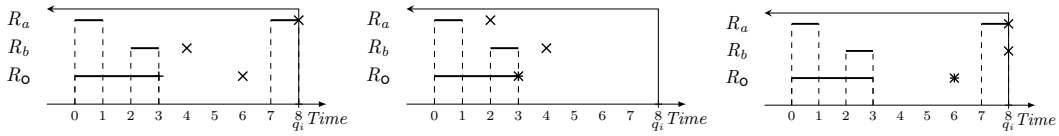


Figure 2: RCT computation for $\mathsf{seq}^{\bullet}(R_a, R_b, R_o)$, with $R_a = \langle I_a, T_a \rangle$, $R_b = \langle I_b, T_b \rangle$ and $R_o = \langle I_o, T_o \rangle$. Horizontal lines correspond to maximal intervals, while crosses and tilted crosses correspond to candidate cut-off and final cut-off points respectively. The arrow to the left denotes the range of the stream for which the computation is performed.

3). Once T_o is computed, the interval lists included in the input RCTs are truncated by T_o (line 4), and the output interval list I_o is computed by applying $\mathsf{RTEC_s}$ on the truncated lists of intervals (line 5).

We assume monotonic input; therefore, at each query-time q_i, the cut-off point of each item of the input stream is set to q_i. Below, we define the process of cut-off point computation for the FVPs defined in terms of the sequence operator or interval manipulation constructs.

Definition 12 (Candidate cut-off point of sequencing). *Let $R_a = \langle I_a, T_a \rangle$ and $R_b = \langle I_b, T_b \rangle$ be the input RCTs of $\mathsf{seq}^{\bullet}(R_a, R_b, R_o)$, and R_o the output RCT. The candidate cut-off point T_o^c of R_o is*

$$T_o^c = \begin{cases} e(i_b) & \text{if } \exists i_a \in I_a, i_b \in I_b. \big(last_i(I_a, T_b, i_a) \wedge s(i_b) < nextPosI(T_a) \\ & \quad \wedge \mathtt{A}(i_a, I_a, I_b) = \{i_b\}\big) \quad \text{(case (a))} \\ s(i_a) - 1 & \text{if } \neg\text{(case (a))} \wedge \exists i_a \in I_a. last_i(I_a, T_b, i_a) \\ T_a & \text{otherwise,} \end{cases} \tag{4}$$

where $last_i(I, T, i)$ is satisfied if i is the last interval of list I that ends before time-point T, $\mathtt{A}$ computes adjacent intervals (see Definition 5), and $nextPosI(T) = T+3$, i.e. $nextPosI(T)$ is the earliest possible time-point allowing an interval $[t_s, t_e)$ with $t_s > T$ and $t_e < nextPosI(T)$.

Example 5 (Candidate cut-off computation for sequencing). *Consider Figure 2-left. The last interval of I_a of R_a before the cut-off point T_b of R_b, i.e. $[0, 1)$, is adjacent to the interval $[2, 3)$ of I_b (case (a) of eq. (4)); consequently T_o^c may be set to 3 as no extension of I_a or I_b can change the output up to T_o^c. In Figure 2-middle, T_o^c is also set to 3 (case (a) of eq. (4)). In Figure 2-right, there is no adjacent interval to the last interval of I_a, i.e. $[7, 8)$, before T_b; therefore, T_o^c is $s([7, 8)) - 1 = 6$.*

For the latest cut-off point, we compute the extension set.

Definition 13 (Extension set of sequencing). *Given $\mathsf{seq}^{\bullet}(R_a, R_b, R_o)$, where $R_a = \langle I_a, T_a \rangle$ and $R_b = \langle I_b, T_b \rangle$, and the candidate cut-off point T_o^c of R_o, the extension set E_s of sequencing*

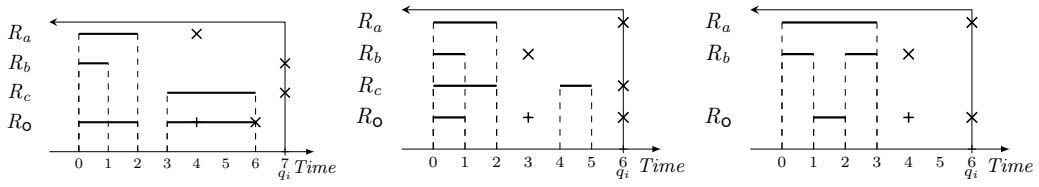


Figure 3: RCT computation for $\mathsf{union\_all}^\bullet([R_a, R_b, R_c], R_o)$ (left), $\mathsf{intersect\_all}^\bullet([R_a, R_b, R_c], R_o)$ (middle) and $\mathsf{relative\_complement\_all}^\bullet(R_a, [R_b], R_o)$ (right).

contains each time-point t in $(T_o^c, \max\{T_a, T_b\}]$ for which, the intervals of any extension of I_a, starting before, or at t, that do not participate in a sequence with an interval of I_b, will not participate in a sequence for any extension of I_b. If $T_o = T_o^c$, then $T_o^c > T_a$, which, by Definition 12, is possible only under case (a). The condition $s(i_b) < nextPosI(T_a)$ in that case ensures that no interval added to I_a after T_a can introduce a new sequence whose start point is at or before T_o^c.

Example 6 (Cut-off computation for sequencing)**.** *Recall that in Figure 2-left T_o^c is set to 3 (Example 5). Since there are no intervals in I_a starting during the period $[4, 6]$, E_s includes time-points $4, 5$ and 6. (Note that the interval $[0, 1)$ of I_a participates in a sequence with an interval of I_b.) E_s does not include time-point 7 because $[7, 8)$ of I_a may participate in a sequence in the future with an interval of I_b. In other words, $E_s = \{4, 5, 6\}$ and thus $T_o = 6$. In Figure 2-middle, we have $T_o^c = 3$. As an interval starting at time-point 4 may be added to I_a in the future, for which an adjacent interval may exist in an extension of I_b, E_s is empty and $T_o = T_o^c = 3$. In Figure 2-right, where $T_o^c = 6$, E_s is empty as a sequence between $[7, 8)$ of I_a and an interval of I_b is possible at a later q_j after q_i; thus $T_o = T_o^c$.*

When the interval manipulation constructs operate on monotonic input they produce monotonic output. Unfortunately, the input to an interval manipulation construct cannot be guaranteed to be monotonic, because such input may be computed by means of the sequence operator (e.g. rule (3')). Thus, similar to sequencing, the evaluation of interval manipulation constructs must be performed on RCTs.

Definition 14 (Candidate cut-off point and extension sets of $\mathsf{iConstruct}^\bullet$)**.** *Let $LR = [R_1, \ldots, R_n]$, with $R_i = \langle I_i, T_i \rangle$ and $i \in [1, n]$, be the list of input RCTs of $\mathsf{iConstruct}^\bullet(LR, R_o)$, and R_o the output RCT. The candidate cut-off point T_o^c of R_o is $T_o^c = \min\{T_i\}$. Moreover, the extension sets E_u, E_i and E_c of, respectively, $\mathsf{union\_all}^\bullet$, $\mathsf{intersect\_all}^\bullet$ and $\mathsf{relative\_complement\_all}^\bullet$ are defined as follows:*

- *E_u contains all time-points t in $(T_o^c, \max\{T_i\}]$ such that all time-points in $[T_o^c, t)$ are contained by at least one interval of the input lists of $\mathsf{union\_all}^\bullet$.*
- *E_i contains all the time-points t in $(T_o^c, \max\{T_i\}]$ such that for all time-points t' in $[T_o^c, t]$ there is at least one interval list in the input of $\mathsf{intersect\_all}^\bullet$ that does not contain t'.*
- *E_c contains all the time-points t in $(T_o^c, \max\{T_i\}]$ such that $[T_o^c, t]$ does not overlap and is not contained by any intervals of the first input interval list of $\mathsf{relative\_complement\_all}^\bullet$.*

Example 7 ($\mathsf{iConstruct}^\bullet$ over RCTs)**.** *Figure 3-left presents an example for $\mathsf{union\_all}^\bullet$, i.e. $\mathsf{union\_all}^\bullet([R_a, R_b, R_c], R_o)$. The candidate cut-off point T_o^c is set to 4, i.e. the minimum cut-off point of the input RCTs. The extension set E_u includes time-points 5 and 6, but not time-point 7 because time-point 6 does not belong to any interval of I_a, I_b or I_c. Thus, $E_u = \{5, 6\}$ and the final cut-off point T_o is set to 6. Figure 3-middle shows*

intersect_all$^{\bullet}([R_a, R_b, R_c], R_o)$. $T_o^c = 3$; *moreover, as there is no possible extension of* I_b *that can change the output up to time-point 6,* $E_i = \{4, 5, 6\}$. *Thus, we can set* T_o *to 6. Figure 3-right illustrates* relative_complement_all$^{\bullet}(R_a, [R_b], R_o)$. *Although* $T_o^c = 4$, *as there is no interval of* I_a, *i.e. the first input interval list of* relative_complement_all$^{\bullet}$, *after or overlapping* T_o^c, *we can extend the cut-off point to 6 (*$E_c = \{5, 6\}$*).*

Proposition 3 (Monotonic reasoning)**.** *The interval list* I_o *computed by means of* holdsFor$^{\bullet}(F = V, \langle I_o, T_o \rangle)$ *at* q_i *will never be revised at a future query-time.*

According to the proposition above, the output of $\mathsf{RTEC}_s^{\mathsf{RCT}}$ at q_i is monotonic up to T_o.

Proposition 4 (Cut-off point optimality)**.** *The cut-off point* T_o *computed by means of* holdsFor$^{\bullet}(F = V, \langle I_o, T_o \rangle)$ *at* q_i *is the latest time-point that guarantees a monotonic output.*

3.3 Stream Reasoning and Bookkeeping

At each query-time q_i, $\mathsf{RTEC}_s^{\mathsf{RCT}}$ performs stream reasoning by operating on (i) a working memory set *WM* containing the items of the input stream in $(q_{i-1}, q_i]$, and (ii) a bookkeeping set *BSet* including the minimal information from past query-times which is necessary for producing correct and monotonic output. The curation of *BSet* is guided by RCTs. In what follows, we present how $\mathsf{RTEC}_s^{\mathsf{RCT}}$ computes the intervals of statically determined FVPs, i.e. FVPs defined in terms of sequencing and/or interval manipulation constructs, as well as how *BSet* is computed.

Like RTEC_s, $\mathsf{RTEC}_s^{\mathsf{RCT}}$ processes FVPs in a bottom-up manner following the dependency graph of the event description, computing and caching their intervals. This way, the intervals of each FVP are computed once and redundant recomputations are avoided. Unlike RTEC_s, $\mathsf{RTEC}_s^{\mathsf{RCT}}$ employs Algorithm 2 to evaluate the rules defining statically determined FVPs. For a statically determined FVP $F=V$, `evalFBody` first retrieves all RCTs occurring through holdsFor$^{\bullet}$ literals in the body of the definition of $F=V$, and stores them in list *RCTs* (line 1). For example, for rule (3'), *RCTs* would contain R_s, R_{sf} and R_a. Then, Algorithm 2 iterates over each interval manipulation construct and sequence operation in the body of the definition of $F=V$. For each operation, `evalFBody` retrieves any previously retained data from *BSet* and merges it with the input RCTs of the operation under consideration (lines 3–5). Next, `evalFBody` invokes `computeRCT` (Algorithm 1) to compute the output RCT of the operation, and updates the *RCTs* list with the result of this operation as it may be used by a subsequent operation in the body of $F=V$ (lines 6–8). For example, in rule (3'), after evaluating the sequence operation, *RCTs* would also contain R_{sa} so that it is available for the intersect_all interval manipulation construct. As a final step, any information that should be cached for future query-times is identified and added to *BSet* (lines 8, 9), according to the policy below.

Definition 15 (Bookkeeping policy)**.** *Consider a sequence operation* seq$^{\bullet}(R_1, R_2, R_o)$ *or an interval manipulation construct* iConstruct$^{\bullet}([R_1, \ldots, R_n], R_o)$. *For each input interval list* I_i *of* R_i *(*$i \in [1, n]$ *or resp.* $i \in [1, 2]$*),* $\mathsf{RTEC}_s^{\mathsf{RCT}}$ *caches in BSet only the intervals that start after or overlap the cut-off point* T_o *of the output RCT* R_o.

Proposition 5 (Soundness and completeness of bookkeeping)**.** *Consider an interval manipulation construct* iConstruct$^{\bullet}([R_1, \ldots, R_n], R_o)$ *or a sequence operation* seq$^{\bullet}(R_1, R_2, R_o)$. *According to the bookkeeping policy of Definition 15,* $\mathsf{RTEC}_s^{\mathsf{RCT}}$ *caches all and only the intervals from* I_i *of* R_i *(*$i \in [1, n]$ *or resp.* $i \in [1, 2]$*) that may be necessary for the computation of the interval manipulation construct or sequence operation at a future query-time.*

Algorithm 2 `evalFBody`$(WM, BSet, F = V)$

```
1: RCTs ← getHoldsForRCTs(FvalV)
2: for op• ∈ body(F = V) do
3:     RetainedData ← BSet.retrieve(op•, F = V)
4:     InpRCTs ← assign(RCTs)
5:     InpRCTs′ ← RetainedData ∪ InpRCTs
6:     OutRCT ← computeRCT(op•, InpRCTs′)
7:     RCTs ← RCTs ∪ OutRCT
8:     DataToRetain ← bkeep(InpRCTs′, OutRCT)
9:     BSet.retain(F = V, op•, DataToRetain)
10: end for
11: return last(RCTs)
```

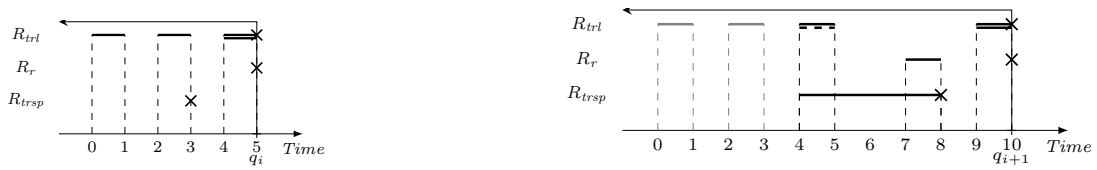


Figure 4: Computing the intervals of *transhipment* at q_i (left) and q_{i+1} (right). Horizontal lines correspond to maximal intervals, double continuous horizontal lines correspond to intervals in *BSet*, double semi-dashed horizontal lines correspond to intervals that were restored from *BSet* and gray horizontal lines correspond to intervals that were discarded.

Example 8 (Stream reasoning)**.** *Recall Example 2 for the definition of transhipment. Figure 4-left shows the RCTs of the 'trawling', 'rendezVous' and 'transhipment' fluents, i.e.* R_{trl}, R_r *and* R_{trsp}, *at* q_i. *The definitions of 'trawling' and 'rendezVous' do not depend on* seq, *and thus their cut-off points are equal to* q_i. *At* q_i, *no interval is computed for 'transhipment' while the cut-off point is set to 3. For bookkeeping, we add the interval* $[4, 5)$ *of 'trawling' in BSet, as it may participate in a sequence at a future query-time. In contrast, the intervals* $[0, 1)$ *and* $[2, 3)$ *of* I_{trl} *do not have to cached. At* q_{i+1} *(Figure 4-right), the interval in BSet is merged with the intervals in WM, i.e. the intervals in* $(q_i, q_{i+1}] = (5, 10]$. *The interval* $[4, 5)$ *of 'trawling' and the interval* $[7, 8)$ *of 'rendezVous' yield a sequence producing 'transhipment' over* $[4, 8)$ *with a cut-off point at 8. Finally, BSet is updated to contain the interval* $[9, 10)$ *of 'trawling', as this interval may participate in a sequence in the future.*

Comparative Discussion. Recall that $\mathsf{RTEC_s}$ relies on a global sliding window of size ω, while the evaluation of seq is stateless. To ensure correctness, ω must be at least as large as the maximum temporal distance between the elements of a sequence; otherwise, early intervals may be discarded before their adjacent intervals appear. This forces $\mathsf{RTEC_s}$ to use large windows for long-distance dependencies. Since these windows are global, they may contain redundant intervals, such as the intervals of FVPs that are not defined in terms of sequence, or the intervals of FVPs defined in terms of shorter sequences. $\mathsf{RTEC_s^{RCT}}$ avoids large global windows — $\mathsf{RTEC_s^{RCT}}$ operates on non-overlapping subsets $(q_{i-1}, q_i]$ of the input stream — by caching in *BSet* the intervals that may affect future reasoning. This way, $\mathsf{RTEC_s^{RCT}}$ is capable of more efficient reasoning than $\mathsf{RTEC_s}$ (see Section 4 for empirical support of this claim).

Proposition 6 (Correctness of $\mathsf{RTEC_s^{RCT}}$)**.** *At each query-time,* $\mathsf{RTEC_s^{RCT}}$ *computes the same*

intervals for $F = V$ *as* $\mathsf{RTEC_s}$*, up to the cut-off point* T_o *of* $\mathsf{holdsFor^\bullet}(F = V, \langle I_o, T_o \rangle)$*, when the window of* $\mathsf{RTEC_s}$ *is large enough so that no sequence is missed.*

Note that, even with sufficiently large windows, $\mathsf{RTEC_s}$ does not guarantee monotonic output; the result of the evaluation of seq may only be confirmed after the latter constituent becomes available, leading to revisions of the past that may propagate through hierarchical FVP definitions.

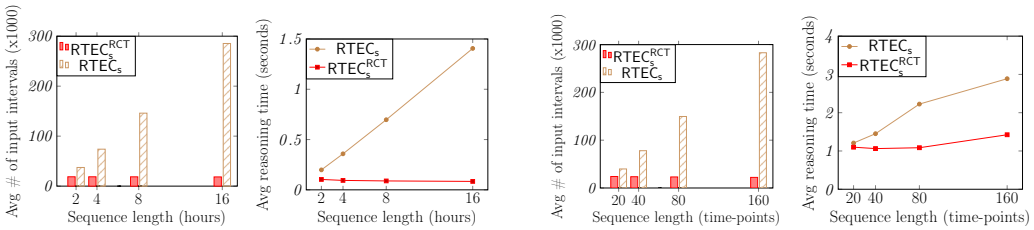
In addition to enabling monotonic output and improving reasoning efficiency, RCTs offer crucial information which is lacking from competing approaches. Consider an RCT $\langle I_o, T_o \rangle$ for an FVP $F=V$, computed at q_i, and assume that the last interval in I_o is $[t_s, t_e)$. According to the definitions of RCT and cut-off point, we know that no interval for $F=V$ will be computed at a future query-time that overlaps the period $[t_e, T_o]$ (e.g. in the left and right diagrams of Figure 2, $[t_e, T_o]$ is $[3, 6]$). In contrast, in other stream reasoning systems with a sequence operator in the language, such as CORE [12], Wayeb [2] and ESPER [17], we can draw no conclusions about the value of a property after the ending point of its last interval. In the empirical analysis (Section 4), we show that the interval $[t_e, T_o]$ can be rather long, indicating that the use of $\mathsf{RTEC_s^{RCT}}$ offers more informed decision-making as opposed to competing approaches.

4 Experimental Evaluation

4.1 Experimental Setup

We conducted an empirical analysis in order to assess the reasoning efficiency of $\mathsf{RTEC_s^{RCT}}$ — in the experiments that follow we report the reasoning times of $\mathsf{RTEC_s^{RCT}}$ and the number of intervals over which reasoning is performed. We evaluated $\mathsf{RTEC_s^{RCT}}$ on Maritime Situational Awareness (MSA), Commercial Fleet Management (CFM) and Multi-Agent Voting (MAV). In MSA, the input stream consists of position signals emitted by vessels, including information about their heading and speed. The task is to compute the maximal intervals of FVPs expressing various types of vessel behaviour, including potentially dangerous and illegal activities. We extended the event description of Pitsikalis et al. [33] with FVPs defined in terms of the sequence operator, in order to capture more accurately the maritime behaviours of interest, such as transhipment (Example 2) and suspicious anchored vessels (Example 3). The extended event description includes 21 FVPs, while the depth of the dependency graph is 5. To stress-test $\mathsf{RTEC_s^{RCT}}$ over long-term sequences, we constructed four synthetic datasets, each simulating trajectories of 1K vessels. Moreover, we evaluated $\mathsf{RTEC_s^{RCT}}$ on two real datasets. The first dataset is publicly available and includes 18M position signals emitted from 5K vessels sailing in the Atlantic Ocean around Brest, France, between October 2015–March 2016 [36]. The second dataset, provided by IMIS (https://www.imisglobal.com), is much larger; it contains 55M position signals from 34K vessels sailing in the European seas between January 1–30, 2016.

In CFM the input stream consists of positional data of commercial vehicles enriched with weather information and proximity to points of interest, such as petrol stations. The task is to compute the maximal intervals of FVPs related to the management and planning of transportation services, such as refuelling opportunities. We extended the event description of Tsilionis et al. [40] with FVPs capturing transitions from safe to dangerous driving, and changes from economic to non-economic driving. The extended event description includes 9 FVPs and the depth of the dependency graph is 3. The dataset is provided by Vodafone Innovus (https://www.vodafoneinnovus.com/) and contains 70M position from 6K vehicles travelling in the European road network, between June 30–August 8, 2018.



(a) MSA: synthetic datasets (1K vessels). (b) Multi-agent voting (1K agents).

Figure 5: Evaluation of $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ and RTEC_s on synthetic data.

As a third use case, we employed a typical MAV protocol [34], which may be summarised as follows: a committee sits and the chair opens the meeting; a member proposes a motion; another member seconds the motion; the members debate the motion; the chair calls for those in favour/against to cast their vote; finally, the motion is carried, or not, according to the standing rules of the committee. In MAV, the task is to compute, for the benefit of the agents and humans overseeing protocol execution, the maximal intervals of FVPs expressing the status of motions, the normative positions of the agents, such as institutional power, permission and obligation, and the sanctions applied to agents performing forbidden actions or not complying with obligations. We extended the event description of Pitt et al. [34] with FVPs facilitating protocol analytics, such as those expressing sequences of different stages of protocol execution. The extended event description includes 7 FVPs and the depth of the dependency graph is 5. We created four synthetic datasets, each containing approx. 1M messages from interactions among 1K agents. The event descriptions for MAV, CFM and MSA, and the instructions to reproduce the experiments, are available with the code of $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$.

Mantenoglou and Artikis [28] presented an experimental comparison of RTEC_s with CORE [12], a state-of-the-art system with highly-optimised reasoning algorithms and formal semantics. The experimental results showed that RTEC_s outperforms CORE, sometimes by orders of magnitude, because it restricts sequence evaluation to adjacent intervals, and exploits caching techniques in hierarchical event descriptions. Given these results, we use RTEC_s as the baseline and compare it against $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$. Our experiments were performed with SWI Prolog 9 and Ubuntu 22, on a machine with a Intel Core i7-8700 CPU@3.20GHz and 16GB of memory.

4.2 Experimental Results

The aim of the first set of experiments was to evaluate $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$, and compare to RTEC_s, on increasing sequence length, i.e. the temporal distance between the elements of a sequence. Towards this, we employed the synthetic datasets — these concern MSA and MAV. Figures 5a and 5b display the results. $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ and RTEC_s performed reasoning at the same query-times, i.e. the step was the same for both systems (e.g. for MSA the step was set to 1 hour). For RTEC_s, we employed the shortest window size ω that avoids false negatives in each sequence length — this way, RTEC_s and $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ compute the same FVP intervals. The left diagrams of Figures 5a and 5b display the average number of intervals over which $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ and RTEC_s perform reasoning at each query-time, i.e. the number of intervals in $(q_i{-}\omega, q_i]$ for RTEC_s, and the number of intervals in $WM \cup BSet$ for $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$. These diagrams show that $\mathsf{RTEC}_\mathsf{s}^\mathsf{RCT}$ reasons over substantially fewer intervals than RTEC_s. This difference becomes more pronounced

Table 1: Evaluation of RTEC_s^{RCT} and RTEC_s on real data for MSA and CFM. 'input' refers to the average number of input intervals and 'time' refers to the average reasoning time.

	Brest		**European seas**		**European roads**	
	input	**time**	**input**	**time**	**input**	**time**
RTEC_s	411K	10.2 sec	N/A	N/A	N/A	N/A
RTEC_s^{RCT}	7.5K	1.3 sec	130K	34.6 sec	101K	2.5 sec

as the sequence length increases. An increase in the sequence length demands an increase of ω for RTEC_s; in contrast, RTEC_s^{RCT} operates on non-overlapping subsets of the stream, and performs correct reasoning with the use of $BSet$, which contains much fewer intervals than those in $(q_i - \omega, q_{i-1}]$. The right diagrams of Figures 5a and 5b display the average reasoning times of RTEC_s^{RCT} and RTEC_s per query-time. These diagrams show that RTEC_s^{RCT} scales better to long-term sequences and outperforms RTEC_s. Apart from having a larger input, RTEC_s has to recompute the FVP intervals in the overlap of successive windows. RTEC_s^{RCT} avoids such redundant recomputations with the use of RCTs, because the intervals of an FVP ending by the corresponding cut-off point are never revised.

The aim of the second set of experiments was to evaluate RTEC_s^{RCT} and RTEC_s on real data. Table 1 presents the results for the Brest dataset (5K vessels), the European seas dataset (34K vessels) and the European roads dataset (6K vehicles). The step was set to 1 hour. In these datasets, the temporal distance between the elements of a sequence often spanned several days; consider e.g. the temporal distance between the intervals during which a vessel is involved in transhipment, and the intervals during which the vessel is anchored (Example 3). This issue affected the performance of RTEC_s since we had to use a large value for ω in order to avoid false negatives. Unfortunately, RTEC_s exhausted the available memory resources in the case of the European seas and the European roads datasets, and thus failed to complete execution. Table 1 confirms that RTEC_s has to operate over a significantly larger number of input intervals in order to produce the same results as RTEC_s^{RCT}, and that RTEC_s^{RCT} has significantly lower reasoning time than RTEC_s. Moreover, Table 1 shows that RTEC_s^{RCT} is capable of supporting MSA and CFM in large geographical areas; the average reasoning time in the European seas dataset is approx. 35 seconds, and 2.5 seconds in the European roads dataset.

Recall that RTEC_s^{RCT} remains monotonic in $[t_e, T_o]$, where t_e is ending point of the last computed interval before the cut-off point T_o, while e.g. automata-based systems such as Wayeb [1, 2], are monotonic until t_e (Section 3.3). In the experiments with the real datasets, the length of $[t_e, T_o]$ ranged between 8 to 37 hours. Thus, RTEC_s^{RCT} may support decision-making for e.g. operational resource allocation for much longer periods than related systems.

5 Summary, Related & Further Work

We presented RTEC_s^{RCT}, an interval-based stream reasoning system. RTEC_s^{RCT} inherits from its predecessor RTEC_s a compositional and associative sequencing operator. RTEC_s^{RCT} extends RTEC_s with RCTs, a mechanism for monotonic output given a monotonic input. Monotonic output is a key enabler of optimal resource allocation, e.g. in MSA and CFM. RTEC_s^{RCT} is also equipped with a bookkeeping policy, guided by RCTs, that identifies the minimal information for correct reasoning. Our empirical, reproducible evaluation on three application domains, using real and synthetic datasets, demonstrated the reasoning efficiency of RTEC_s^{RCT}.

Sequencing has long been recognised as a fundamental construct in Complex Event Recognition (CER) [14]. Most CER systems operate on monotonic input and are automata-based or tree-based. Unfortunately, it has been observed [21] that the majority of these systems, such as ESPER [17], lack a formal, declarative semantics, or exhibit unintuitive semantics (e.g. sequencing is non-associative [15]). Two exceptions are CORE [12] and Wayeb [1, 2]. A comparative analysis [28] showed that restricting sequence evaluation to adjacent intervals (Definition 5) allows $\mathsf{RTEC_s}$ to have lower complexity than CORE and Wayeb. Moreover, automata-based CER systems guarantee monotonic output only up to the last reported result. In contrast, RCTs allow $\mathsf{RTEC_s^{RCT}}$ to be monotonic long after the last reported interval — our empirical analysis showed that $\mathsf{RTEC_s^{RCT}}$ was monotonic for 8-37 hours after the last computed interval.

Although automata-based systems typically support compositional definitions, they do not allow for hierarchical definitions with reusable intermediate results, thus performing unnecessary recomputations of shared subdefinitions. $\mathsf{RTEC_s^{RCT}}$ supports arbitrary hierarchies of FVPs defined in terms of sequencing, interval manipulation constructs, and the common-sense law of inertia. The event description for MSA e.g. includes a hierarchy of 21 simple and statically determined FVPs. Each FVP is accompanied by its RCT, indicating the FVP intervals that may be safely cached, thus avoiding redundant recomputations while preserving monotonicity.

Liu et al. [26] proposed 'Partial Order Guarantees' (POG)s to enable monotonic output over non-monotonic input. POGs indicate the future non-occurrence of an event type. The assumption is that it is possible to insert POGs at each stream source. Unfortunately, such as assumption is rarely realistic. Moreover, Liu et al. [26] perform point-based, as opposed to interval-based reasoning — the issues of such reasoning are well-documented [47]. Producing monotonic output with sequencing over non-monotonic input is a direction for further work.

Several logic-based stream reasoning systems, with a formal semantics, have been proposed [11]. LARS [8] is a stream reasoning language expressing interval-based rules. LARS-based reasoners [43, 16, 9, 7], however, support fragments of LARS that cannot express interval derivations. StreamMill [25] adopts a minimal extension of SQL for stream reasoning. MeTeoR [45] extends a fragment of DatalogMTL [44] with windowing. Wang et al. [46] extended the 'magic sets' rewriting technique to the temporal setting of DatalogMTL. Ronca et al. [37] studied the properties of streaming algorithms for a temporal extension of Datalog. Similar to MeTeoR and StreamMill, this extension of Datalog does not support negation. Thus, these approaches cannot express the event descriptions of $\mathsf{RTEC_s^{RCT}}$. Temporal Vadalog [10] is a reasoning system supporting various fragments of DatalogMTL and stratified negation. None of the aforementioned approaches has explicit support for sequencing. (DDLog [38], DCDatalog [49] and BigDatalog [39] cannot be directly employed for temporal reasoning.)

D^{2}IA extends Flink with temporal operators [6], but, contrary to $\mathsf{RTEC_s^{RCT}}$, does not support negation on durative phenomena. TPStream computes durative activities based on instantaneous events, and then matches temporal patterns over the derived activities [23]. In contrast to $\mathsf{RTEC_s^{RCT}}$, TPStream cannot combine multiple event types in the definition of a durative activity. Moreover, D^{2}IA and TPStream do not allow background knowledge. ETALIS is a CER engine that supports patterns with durative activities and background knowledge [3]. ETALIS does not allow for the explicit representation of time. Metric Compass Logic is a metric extension of the Halpern-Shoham logic [22] that has been used for inteval-based temporal event pattern matching [42]. This logic, however, is propositional, and thus does not capture the relational rules that may be expressed by $\mathsf{RTEC_s^{RCT}}$.

$\mathsf{RTEC_s^{RCT}}$ is orthogonal to the remaining extensions of RTEC that have been proposed in the literature; e.g. Mantenoglou and Artikis [28] showed that the extension of RTEC supporting Allen's interval algebra does not offer associative sequencing. Several techniques have been pro-

posed for optimising interval-based reasoning with the Event Calculus — [13], [30] and [18] are but a few examples. The optimisation techniques of RTEC, which are inherited by $\mathsf{RTEC}_{\mathsf{s}}^{\mathsf{RCT}}$, allow it to outperform competing approaches [29]. As a direction for further work, we aim to extend the tensor-based implementation of the Event Calculus [41] with interval-based reasoning, including a support for a sequence operator, in order to take advantage of modern hardware.

Acknowledgements

This work was supported by the EU-funded CREXDATA project (No 101092749) and by the University of Piraeus Research Center.

References

[1] Elias Alevizos, Alexander Artikis, and Georgios Paliouras. Wayeb: a tool for complex event forecasting. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR*, EPiC Series in Computing, pages 26–35. EasyChair, 2018.

[2] Elias Alevizos, Alexander Artikis, and Georgios Paliouras. Complex event recognition with symbolic register transducers. *Proc. VLDB Endow.*, 17(11):3165–3177, 2024.

[3] Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic Web*, 3(4):397–407, 2012.

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

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

[6] Ahmed Awad, Riccardo Tommasini, Samuele Langhi, Mahmoud Kamel, Emanuele Della Valle, and Sherif Sakr. D^2ia: User-defined interval analytics on distributed streams. *Inf. Syst.*, 104:101679, 2022.

[7] Hamid R. Bazoobandi, Harald Beck, and Jacopo Urbani. Expressive stream reasoning with laser. In *ISWC*, volume 10587, pages 87–103, 2017.

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

[9] Harald Beck, Thomas Eiter, and Christian Folie. Ticker: A system for incremental asp-based stream reasoning. *Theory Pract. Log. Program.*, 17(5-6):744–763, 2017.

[10] Luigi Bellomarini, Livia Blasi, Markus Nissl, and Emanuel Sallinger. The temporal vadalog system: Temporal datalog-based reasoning. *Theory Pract. Log. Program.*, 25(2):168–196, 2025.

[11] Pieter Bonte, Jean-Paul Calbimonte, Daniel de Leng, Daniele Dell'Aglio, Emanuele Della Valle, Thomas Eiter, Federico Giannini, Fredrik Heintz, Konstantin Schekotihin, Danh Le Phuoc, Alessandra Mileo, Patrik Schneider, Riccardo Tommasini, Jacopo Urbani, and Giacomo Ziffer. Grounding stream reasoning research. *TGDK*, 2(1):2:1–2:47, 2024.

[12] Marco Bucchi, Alejandro Grez, Andrés Quintana, Cristian Riveros, and Stijn Vansummeren. CORE: a complex event recognition engine. *Proc. VLDB Endow.*, 15(9):1951–1964, 2022.

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

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

[15] Alan J. Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker M. White. Towards expressive publish/subscribe systems. In Yannis E. Ioannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthes, Michael Hatzopoulos, Klemens Böhm, Alfons Kemper, Torsten Grust,

and Christian Böhm, editors, *EDBT*, volume 3896 of *Lecture Notes in Computer Science*, pages 627–644. Springer, 2006.

[16] Thomas Eiter, Paul Ogris, and Konstantin Schekotihin. A distributed approach to LARS stream reasoning (system paper). *Theory Pract. Log. Program.*, 19(5-6):974–989, 2019.

[17] EsperTech. Esper enterprise edition website. http://www.espertech.com/, Retrieved February 10 2026.

[18] Nicola Falcionelli, Paolo Sernani, Albert Brugués de la Torre, Dagmawi Neway Mekuria, Davide Calvaresi, Michael Schumacher, Aldo Franco Dragoni, and Stefano Bromuri. Indexing the event calculus: Towards practical human-readable personal health systems. *Artif. Intell. Medicine*, 96:154–166, 2019.

[19] Antony Galton and Juan Carlos Augusto. Two approaches to event definition. In *DEXA*, volume 2453, pages 547–556, 2002.

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

[21] Alejandro Grez, Cristian Riveros, Martín Ugarte, and Stijn Vansummeren. A formal framework for complex event recognition. *ACM Trans. Database Syst.*, 46(4), December 2021.

[22] Joseph Y. Halpern and Yoav Shoham. A propositional modal logic of time intervals. *J. ACM*, 38(4):935–962, 1991.

[23] Michael Körber, Nikolaus Glombiewski, Andreas Morgen, and Bernhard Seeger. Tpstream: low-latency and high-throughput temporal pattern matching on event streams. *Distributed Parallel Databases*, 39(2):361–412, 2021.

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

[25] Nikolay Laptev, Barzan Mozafari, Hamid Mousavi, Hetal Thakkar, Haixun Wang, Kai Zeng, and Carlo Zaniolo. Extending relational query languages for data streams. In *Data Stream Management - Processing High-Speed Data Streams*, Data-Centric Systems and Applications, pages 361–386. Springer, 2016.

[26] Mo Liu, Ming Li, Denis Golovnya, Elke A. Rundensteiner, and Kajal T. Claypool. Sequence pattern query processing over out-of-order event streams. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 784–795. IEEE Computer Society, 2009.

[27] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

[28] Periklis Mantenoglou and Alexander Artikis. Sequencing in the run-time event calculus. In Inês Lynce, Nello Murano, Mauro Vallati, Serena Villata, Federico Chesani, Michela Milano, Andrea Omicini, and Mehdi Dastani, editors, *ECAI*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pages 1607–1614. IOS Press, 2025.

[29] Periklis Mantenoglou, Manolis Pitsikalis, and Alexander Artikis. Reasoning over streams of events with delayed effects. *J. Artif. Intell. Res.*, 84, 2025.

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

[31] Christos Nomikos, Panos Rondogiannis, and Manolis Gergatsoulis. Temporal stratification tests for linear and branching-time deductive databases. *Theor. Comput. Sci.*, 342(2-3):382–415, 2005.

[32] 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.

[33] Manolis Pitsikalis, Alexander Artikis, Richard Dreo, Cyril Ray, Elena Camossi, and Anne-Laure Jousselme. Composite event recognition for maritime monitoring. In *ACM DEBS*, pages 163–174, 2019.

[34] Jeremy Pitt, Lloyd Kamara, Marek Sergot, and Alexander Artikis. Voting in multi-agent systems. *Comput. J.*, 49(2):156–170, March 2006.

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

[36] Cyril Ray, Richard Dréo, Elena Camossi, Anne-Laure Jousselme, and Clément Iphar. Heterogeneous integrated dataset for maritime intelligence, surveillance, and reconnaissance. *Data in Brief*, 25:104141, 2019.

[37] Alessandro Ronca, Mark Kaminski, Bernardo Cuenca Grau, and Ian Horrocks. The delay and window size problems in rule-based stream reasoning. *Artif. Intell.*, 306:103668, 2022.

[38] Leonid Ryzhyk and Mihai Budiu. Differential datalog. *Datalog*, 2:4–5, 2019.

[39] Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. Big data analytics with datalog queries on spark. In *SIGMOD*, pages 1135–1149, 2016.

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

[41] Efthimis Tsilionis, Alexander Artikis, and Georgios Paliouras. A Tensor-Based Probabilistic Event Calculus. In *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 643–652, 10 2025.

[42] Dogan Ulus, Thomas Ferrère, Eugene Asarin, Dejan Nickovic, and Oded Maler. Elements of timed pattern matching. *ACM Trans. Embed. Comput. Syst.*, 23(4):59:1–59:45, 2024.

[43] Jacopo Urbani, Markus Krötzsch, and Thomas Eiter. Chasing streams with existential rules. In *KR*, 2022.

[44] Przemyslaw Andrzej Walega, Mark Kaminski, and Bernardo Cuenca Grau. Reasoning over streaming data in metric temporal datalog. In *AAAI*, 2019.

[45] Przemyslaw Andrzej Walega, Mark Kaminski, Dingmin Wang, and Bernardo Cuenca Grau. Stream reasoning with DatalogMTL. *J. Web Semant.*, 76, 2023.

[46] Shaoyu Wang, Kaiyue Zhao, Dongliang Wei, Przemyslaw Andrzej Walega, Dingmin Wang, Hongming Cai, and Pan Hu. Goal-driven reasoning in datalogmtl with magic sets. In Toby Walsh, Julie Shah, and Zico Kolter, editors, *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 15203–15211. AAAI Press, 2025.

[47] Walker White, Mirek Riedewald, Johannes Gehrke, and Alan Demers. What is “next” in event processing? In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, page 263–272, Beijing China, June 2007. ACM.

[48] Walker M. White, Mirek Riedewald, Johannes Gehrke, and Alan J. Demers. What is "next" in event processing? In *PODS*, pages 263–272, 2007.

[49] Jiacheng Wu, Jin Wang, and Carlo Zaniolo. Optimizing parallel recursive datalog evaluation on multicore machines. In *SIGMOD*, 2022.