Reasoning over Streams of Events with Delayed Effects

PERIKLIS MANTENOGLOU*, NCSR "Demokritos", Greece
MANOLIS PITSIKALIS, NCSR "Demokritos", Greece
ALEXANDER ARTIKIS, NCSR "Demokritos" & University of Piraeus, Greece

In streaming applications, it is often required to detect situations of interest, by means of temporal pattern matching, with minimal latency. In the maritime domain, e.g., where it is crucial to prevent activities that are harmful to the environment, we need to report illegal fishing activities, based on streams of low-level vessel actions, as soon as possible. Streams often include events with delayed effects. In multi-agent voting protocols, e.g., a proposed motion may be seconded at the latest by some time in the future. In simulations of biological systems, a signal may lead to the deactivation of the functions of a gene after a time delay. We propose a formal computational framework that handles streams including events with delayed effects. We present the syntax, semantics and reasoning algorithms of our proposed framework, and demonstrate its correctness and complexity. Furthermore, we present a reproducible analysis on large synthetic and real data streams, from the fields of composite event recognition, multi-agent systems and biological feedback processes, and compare the efficiency of our approach with state-of-the-art systems that can perform stream reasoning in these domains. Our results demonstrate that our framework is capable of reasoning over very large streams, including events with delayed effects, while outperforming the state-of-the-art, often by orders of magnitude.

JAIR Associate Editor: Kuldeep Meel

JAIR Reference Format:
Periklis Mantenoglou, Manolis Pitsikalis, and Alexander Artikis. 2025. Reasoning over Streams of Events with Delayed Effects. *Journal of Artificial Intelligence Research* 84, Article 14 (October 2025), 37 pages. DOI: 10.1613/jair.1.17886

1 Introduction

In contemporary applications, large streams of events, i.e., low-level, symbolic, time-stamped data, are generated continuously and with high velocity. In order to make sense of such data and react to critical situations, there is a need for frameworks that perform temporal pattern matching over event streams and report instances of pattern satisfaction with minimal latency. Streaming applications include, e.g., the monitoring of robot behaviour with respect to safety constraints (Tiger and Heintz 2020), the identification of security incidents based on network traffic logs that are generated in real-time (Armbrust et al. 2018), the computation of the values of biological variables in simulations of feedback processes (Srinivasan et al. 2022), and the detection of (illegal) fishing activities on streams of vessel actions, derived by pre-processing vessel position signals (Pitsikalis et al. 2019).

Streaming applications introduce several challenges (Bonte et al. 2024; Dell'Aglio et al. 2017). Temporal pattern matching over event streams requires an expressive specification language, in order to model the patterns describing the situations of interest, which often involve multiple spatio-temporal constraints and background

*Corresponding Author.

Authors' Contact Information: Periklis Mantenoglou, ORCID: 0009-0002-3275-1522, pmantenoglou@iit.demokritos.gr, NCSR "Demokritos", Greece; Manolis Pitsikalis, ORCID: 0000-0003-2959-2022, manospits@iit.demokritos.gr, NCSR "Demokritos", Greece; Alexander Artikis, ORCID: 0000-0001-6899-4599, a.artikis@unipi.gr, NCSR "Demokritos" & University of Piraeus, Greece.

DOI: 10.1613/jair.1.17886

knowledge. In the maritime domain, e.g., we may need to detect instances of 'trawling', i.e., a common type of fishing activity where a vessel performs several consecutive turns while sailing in a fishing area within certain speed limits. Moreover, in order to handle large, high-velocity and unbounded event streams, we need highly efficient reasoning algorithms; these typically employ sliding windows over the stream (Verwiebe et al. 2023). To ensure correctness, the detected instances of pattern satisfaction must be the same as they would have been if the entire history of events was available (Ronca, Kaminski, Grau, and Horrocks 2022; Ronca, Kaminski, Grau, Motik, et al. 2018).

Furthermore, streaming applications often include events with delayed effects. In multi-agent e-commerce protocols, e.g., establishing a contract obliges the consumer to pay the agreed price, and the merchant to deliver the goods, within a specified time (Chopra et al. 2020). In simulations of biological feedback processes, the activation of a gene may affect the concentration of a protein after a time delay (Srinivasan et al. 2022). Consider also composite event recognition for maritime situational awareness (Giatrakos et al. 2020), where a trawling activity is said to end some minutes after a 'change in heading' event, provided that no other event of this type has taken place in the meantime. The formalisms that have been proposed for handling events with delayed effects (Chopra et al. 2020; Demolombe 2014; Hindriks and Riemsdijk 2013; Karlsson et al. 1998; Marín and Sartor 1999; Miller and Shanahan 2002), do not scale to streaming applications, where continuous queries over evolving data streams and complex temporal specifications need to be computed with minimal latency.

To address this issue, we propose RTEC$^{\rightarrow}$, a formal computational framework that handles streams including events with delayed effects. We developed RTEC$^{\rightarrow}$ by extending the 'Run-Time Event Calculus' (RTEC), a logic-based framework that is optimised for reasoning over event streams using temporal windows (Artikis, Sergot, and Paliouras 2015; Mantenoglou, Kelesis, et al. 2023; Mantenoglou, Pitsikalis, et al. 2022). These features are typically absent from formal, automata-based approaches (Bucchi et al. 2022; Grez, Riveros, Ugarte, and Vansummeren 2021). Moreover, RTEC supports out-of-order streams, while automata-based systems typically need to order the input stream before processing. RTEC expresses durative activities by means of the law of inertia, as specified in the Event Calculus, i.e., a logic programming formalism for reasoning about the effects of events over time (Kowalski and Sergot 1986). In contrast, frameworks for temporal pattern matching with a time-point-based semantics complicate the representation of durative activities and produce ambiguous results (Paschke and Bichler 2008; White et al. 2007). Though there are several frameworks that use the Event Calculus for temporal reasoning (Arias, Carro, Chen, et al. 2022; Falcionelli et al. 2019; Montali et al. 2013), we chose to focus on RTEC, because it has proven to be highly efficient in streaming applications, such as the monitoring of multi-agent systems, and composite event recognition for maritime situational awareness, outperforming the state-of-the-art (Mantenoglou, Kelesis, et al. 2023; Mantenoglou, Pitsikalis, et al. 2022).

The contributions of this work may be summarised as follows. First, we present RTEC$^{\rightarrow}$, i.e., an open-source[1], formal computational framework for reasoning over streams of events with delayed effects. We present the language of RTEC$^{\rightarrow}$ and its semantics. Second, we present a set of novel reasoning algorithms. At compile-time, we establish the optimal processing order of a given temporal specification. At run-time, we reason over events with delayed effects incrementally, caching intermediate computations, and thus avoiding re-computations. Moreover, since stream reasoning systems have to operate over sliding windows, in order to handle the volume and velocity of the incoming data, we identify the minimal information that should be transferred between such windows. Third, we prove the correctness of RTEC$^{\rightarrow}$ and present its complexity in the task of detecting the time periods during which domain properties hold, which illustrates the benefits of the proposed reasoning algorithms. Fourth, we present a comprehensive, *reproducible* empirical evaluation based on large synthetic and real data streams, and complex temporal specifications, including a comparison with related systems. Our evaluation demonstrates that RTEC$^{\rightarrow}$ reasons efficiently over streams of events with delayed effects, verifying the

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

complexity analysis, and outperforms the state-of-the-art by orders of magnitude. Instructions for reproducing our experiments are available in Appendix D.

2 Event Calculus for Run-Time Reasoning

The 'Run-Time Event Calculus' (RTEC) is a logic programming implementation of the Event Calculus, i.e., a formalism for representing and reasoning about events and their effects (Kowalski and Sergot 1986).

Representation. The time model is linear with integer time-points. Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. The specification of a domain of interest in RTEC 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 RTEC): A domain in RTEC 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 the 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 express 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 $a_1, a_2, \ldots, a_n$, which may be variables or elements of $\mathcal{O}$, where n is the arity of e. ■

Definition 3 (Fluent-Value Pair): 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 $a_1, a_2, \ldots, a_n$, 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 Fluent-Value Pairs in NetBill): NetBill is a multi-agent protocol for exchanging digital goods (Artikis and Sergot 2010). An e-commerce domain in RTEC that uses the NetBill protocol, i.e., $D=(\mathcal{O}_N, \mathcal{E}_N, \mathcal{F}_N, \mathcal{V}_N)$, may include events '*present_quote*(M, C, G, P)' and '*accept_quote*(C, M, G)', where *present_quote*(M, C, G, P) expresses that a merchant M presents a price P for goods G to a consumer C, while *accept_quote*(C, M, G) states that consumer C accepts the offer of merchant M about goods G. *present_quote* and *accept_quote* are elements of $\mathcal{E}_N$, and M, C, G and P are variables. The domain may also include fluent '*quote*(M, C, G)', where *quote* $\in \mathcal{F}_N$, which expresses that an offer from merchant M to consumer C about goods G is standing. The possible values of *quote*(M, C, G), i.e., the elements of set $\mathcal{V}_N^{quote}$, may be 'true', expressing that a quote is in effect and not close to expiring, '*expiring*', stating that the quote is active but it is about to expire, and 'false', expressing that the quote is not is effect. □

RTEC expresses event occurrences and fluent values over time using a set of domain-independent predicates.

Definition 4 (RTEC Predicates): RTEC employs the following domain-independent predicates:

- happensAt(E, T): event E occurs at time-point T.
- initiatedAt$(F=V, T)$: a time period during which fluent F has value V is initiated at time-point T.
- terminatedAt$(F=V, T)$: a time period during which fluent F has value V is terminated at time-point T.
- holdsAt$(F=V, T)$: fluent F has value V at time-point T.
- holdsFor$(F=V, I)$: fluent F has value V continuously during the maximal interval in list I. ■

A formalisation in RTEC consists of domain-independent and domain-specific rules. The former set of rules derive the list of maximal intervals I in holdsFor$(F=V, I)$, as well as the time-points T where holdsAt$(F=V, T)$ is true. The former set of rules derive the list of maximal intervals I in holdsFor$(F=V, I)$, as well as the time-points T where holdsAt$(F=V, T)$ is true. The set of domain-specific rules, which expresses the relations between the entities of a domain, is called *event description*.

Definition 5 (Event Description in RTEC): An event description in RTEC is a set of:

(1) Ground facts in the form of happensAt(E, T), expressing a stream of event instances. happensAt(E, T) denotes that event E occurs at time-point T.

(2) Rules with initiatedAt($F = V$, T) or terminatedAt($F = V$, T) as their head, expressing the effects of events on fluents. initiatedAt($F = V$, T) (resp. terminatedAt($F = V$, T)) denotes that a time period during which a fluent F has value V is initiated (terminated) at time-point T. ■

According to Definition 5, the specification of the temporal patterns of an application is performed by means of rules with initiatedAt($F = V$, T) or terminatedAt($F = V$, T) as their head.

Definition 6 (Syntax): The rules defining initiatedAt predicates have the following syntax:

$$\begin{aligned} &\text{initiatedAt}(F = V, T) \leftarrow \\ &\quad \text{happensAt}(E_1, T)[[, \\ &\quad [\text{not}]\ \text{happensAt}(E_2, T), \ \ldots, [\text{not}]\ \text{happensAt}(E_i, T), \\ &\quad [\text{not}]\ \text{holdsAt}(F_1 = V_1, T), \ \ldots, [\text{not}]\ \text{holdsAt}(F_k = V_k, T)]]. \end{aligned} \tag{1}$$

We use the standard notation for logic programming (Lloyd 1987), where, e.g., ',' denotes conjunction and '←' denotes implication. The first body literal of a rule defining an initiatedAt predicate is a positive happensAt predicate; this is followed by a possibly empty set of positive/negative happensAt and holdsAt predicates denoted by '[[ ]]'. 'not' expresses negation-by-failure (Clark 1978), while '[not]' denotes that 'not' is optional. All predicates are evaluated on the same time-point, i.e., rules of the above form express the immediate effects of events. terminatedAt rules have the same form. ■

According to the common-sense law of inertia, $F = V$ holds at a time-point T, i.e., holdsAt($F = V$, T) is true, if $F = V$ has been 'initiated' at some earlier time-point, and not 'broken' in the meantime. We formalise the instances where a fluent-value pair (FVP), i.e., $F = V$, is 'broken' as follows:

Definition 7 ('Broken' Fluent-Value Pair): An FVP $F = V$ is said to be 'broken' at time-point T if we have terminatedAt($F = V$, T) or initiatedAt($F = V'$, T), for some $V' \neq V$. ■

According to Definition 7, a fluent cannot have more than one value at any time; initiatedAt($F = V'$, T) implies that $F = V$ is 'broken', for all $V \neq V'$. Moreover, a fluent may not have a value at some time-point(s). It is not the same, e.g., to initiate $F = \text{false}$ and to terminate $F = \text{true}$: the former implies, but is not implied by, the latter.

Example 2 (Quotes): Consider the following example formalisation from NetBill.

$$\begin{aligned} &\text{initiatedAt}(\mathit{quote}(M, C, G) = \text{true}, T) \leftarrow \\ &\quad \text{happensAt}(\mathit{present\_quote}(M, C, G, P), T). \end{aligned} \tag{2}$$

$$\begin{aligned} &\text{initiatedAt}(\mathit{quote}(M, C, G) = \text{false}, T) \leftarrow \\ &\quad \text{happensAt}(\mathit{accept\_quote}(C, M, G), T). \end{aligned} \tag{3}$$

According to rule (2), a $\mathit{quote}(M, C, G)$ is initiated when a merchant M presents a price P for goods G to a consumer C. The *quote* is 'broken' when it is accepted by the consumer (see rule (3)). *quote* is used in the specification of the effects of *accept_quote*; an event $\mathit{accept\_quote}(C, M, G)$ at time-point T may lead to a contract between C and M about G if $\mathit{quote}(M, C, G) = \text{true}$ holds at T. □

The satisfaction of initiatedAt($F = V$, T) does not necessarily imply that $F \neq V$ at T. Similarly, the satisfaction of terminatedAt($F = V$, T) does not necessarily imply that $F = V$ at T. Assume that $F = V$ is initiated at time-points *5* and *25* and terminated at time-points *35* and *50*, and at no other time-points. In that case, $F = V$ holds at all time-points T such that $5 < T \leq 35$.

Semantics. The initiation/termination of an FVP may depend on other FVPs. These dependencies may be modelled in a dependency graph.

Definition 8 (Dependency Graph in RTEC): The dependency graph of an event description in RTEC is a directed graph $G = (\mathcal{V}, \mathcal{E})$, where:

(1) $\mathcal{V}$ contains one vertex $v_{F=V}$ for each FVP $F = V$.

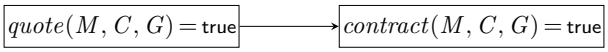

Fig. 1: Dependency graph corresponding to a fragment of the event description for NetBill.

(2) $\mathcal{E}$ contains edge $(v_{F=V}, v_{F'=V'})$ iff there is a rule with initiatedAt$(F'=V', T)$ or terminatedAt$(F'=V', T)$ in its head having holdsAt$(F=V, T)$ as one of its conditions. ■

A dependency graph in RTEC may or may not contain cycles. To simplify the presentation, in this section, we will restrict our attention to acyclic graphs. The treatment of cycles will be discussed in the following sections.

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

Definition 9 (Vertex/FVP Level): Given a directed acyclic graph, the level of a 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.

The level of an FVP $F=V$ is equal to the level of vertex $v_{F=V}$ in the dependency graph. ■

In general, RTEC restricts attention to event descriptions where FVPs with the same fluent have the same level.

Figure 1 presents the dependency graph corresponding to a fragment of the event description for NetBill. $contract(M, C, G) = \text{true}$ expresses that there is an active contract between merchant M and consumer C about goods G; this is initiated by the occurrence of an $accept\_quote(C, M, G)$ action at a time when $quote(M, C, G) = \text{true}$ holds. According to Figure 1, FVP $quote(M, C, G) = \text{true}$ has level 1, and FVP $contract(M, C, G) = \text{true}$ has level 2.

Proposition 1 (Semantics of RTEC): An event description in RTEC is a locally stratified logic program (Przymusinski 1988). ▲

A stratification of an event description may be constructed as follows. The first stratum contains all groundings of happensAt(E, T) atoms, i.e., all atoms derived by substituting E with a ground event and T with a ground time-point. The remaining strata may be formed following the FVP levels in ascending order. For the event description whose dependency graph is shown in Figure 1, e.g., the groundings of RTEC predicates with FVP $quote(M, C, G) = \text{true}$, such as initiatedAt$(quote(M, C, G) = \text{true}, T)$, are placed in lower stratum than the groundings of RTEC predicates with FVP $contract(M, C, G) = \text{true}$.

Reasoning. The key reasoning task of RTEC is the computation of holdsFor$(F=V, I)$, i.e., the list of maximal intervals I during which an FVP $F=V$ of an event description holds continuously. To achieve this, RTEC computes the 'initiation points' of $F=V$ by evaluating, at each time-point of the input, whether the conditions of an initiatedAt rule for $F=V$ are satisfied. Then, RTEC computes all time-points T at which $F=V$ is 'broken' (see Definition 7), by evaluating whether a terminatedAt rule for $F=V$ or an initiatedAt rule for $F=V'$, for some $V' \neq V$, is satisfied at T. These are the 'termination points' of $F=V$. Subsequently, RTEC constructs the list of maximal intervals of $F=V$ by pairing each initiation point T_s of $F=V$ with the first termination point T_f after T_s, ignoring every intermediate initiation point between T_s and T_f. RTEC maintains initiation points, termination points and maximal intervals in temporally sorted lists, facilitating efficient maximal interval computation. See (Artikis, Sergot, and Paliouras 2015) for a complexity analysis of RTEC.

RTEC processes FVPs in a bottom-up manner, computing and caching their intervals level-by-level (Definition 9). 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. Moreover, RTEC performs continuous query processing to compute the maximal intervals of FVPs. At each 'query time' q_i, the events that fall within a specified sliding window $(q_i-\omega, q_i]$ are taken into consideration, while all other events are discarded/'forgotten'. This ensures that the cost of reasoning depends on the size of the window ω and not on the complete stream. In the case that events arrive at RTEC out-of-order, e.g., due to variable time lags in network transmissions, it is preferable to make

ω longer than the 'step', i.e., $q_i - q_{i-1}$, in which case successive windows are overlapping. This way, we may compute, at q_i, the effects of events that took place in $(q_i - \omega, q_{i-1}]$, but arrived after q_{i-1}. The overlap between successive windows should be sufficiently long to capture events that arrive with a time lag, ensuring correct maximal interval computation. At the same time, it should be short enough to minimise re-computations of intervals within the overlap region. Since the temporal extent of event lags varies across application domains, the optimal overlap is domain-specific, and may be determined manually by a domain expert or learnt from data. Tsilionis et al. (2022) developed a reasoning algorithm that minimises the computations within the overlap region of successive windows, while maintaining correctness, thus facilitating the use of longer windows to capture events arriving after a time lag.

3 Delayed Effects of Events

Temporal specifications often include events with delayed effects. In simulations of biological systems, a signal may dictate that the functions of a gene will be switched off after a specified time delay (Srinivasan et al. 2022). In multi-agent voting protocols, the chair may be permitted to close the ballot of a motion only after a specified time since the start of the voting procedure (Pitt et al. 2006). In e-commerce, a contract dictates a set of normative positions with deadlines for the corresponding parties (Hindriks and Riemsdijk 2013). In the legal domain, a law may become applicable after a specified time period has passed from its publication (Marín and Sartor 1999). In maritime situational awareness, the navigational status of a passenger ship is expected to change from 'under way' to 'moored' no later than a specified time after the ship started sailing (Montali et al. 2013). To capture such phenomena, we present RTEC$^{\rightarrow}$, an extension of RTEC that supports events with delayed effects.

The delayed effects of an event should not be confused with the potential delay with which an event may arrive at RTEC$^{\rightarrow}$ (due to, say, transmission issues). These are orthogonal types of delay; e.g., an event may arrive with a delay and have immediate effects, or an event may arrive on time and have delayed effects. To avoid confusion, we use 'lag' to express a delay in the arrival of an event.

3.1 Representation

A key feature of RTEC$^{\rightarrow}$ is the support of events that may change the values of fluents in the future.

Definition 10 (Event Description in RTEC$^{\rightarrow}$): An event description in RTEC$^{\rightarrow}$ extends RTEC event descriptions with the following types of facts:

- $\mathsf{fi}(F=V, F=V', R)$, where $F=V$ and $F=V'$ are FVPs with the same fluent, $V \neq V'$, and R is a positive integer, expressing that the initiation of $F=V$ at a time-point T leads to the ***f**uture **i**nitiation* (fi) of $F=V'$ at time-point $T+R$, provided that $F=V$ is not 'broken' between T and $T+R$. We call R the *delay* of the effects of the event(s) initiating $F=V$. Moreover, if $F=V$ is 'broken' between T and $T+R$, then we say that the future initiation of $F=V'$ is *cancelled*.
- $\mathsf{p}(F=V)$, expressing that the future initiation specified by $\mathsf{fi}(F=V, F=V', R)$ may be *postponed*. For instance, assuming an initiation of $F=V$ at T, a re-initiation of $F=V$ at some time-point T', where $T<T'<T+R$, moves the future initiation of $F=V'$ from $T+R$ to $T'+R$, provided that $F=V$ is not 'broken' between T' and $T'+R$. ■

Definition 10, which extends Definition 5, presents how one may use the language of RTEC$^{\rightarrow}$ to express a temporal specification.

According to Definition 10, in addition to immediate effects, events may have *delayed effects*. Consider a set of events $E_{F=V}$ that appear with positive happensAt predicates in the body of an initiatedAt rule for $F=V$ (rule (1)). An immediate effect of the events in $E_{F=V}$ is the initiation of $F=V$, provided that all other conditions, if any, of the initiatedAt rule are satisfied. If the event description includes $\mathsf{fi}(F=V, F=V', R)$, then the events in $E_{F=V}$ may additionally have delayed effects, in the sense that $F=V'$, where $V' \neq V$, may be initiated in the future.

Example 3 (Expiring Quotes): Recall rules (2) and (3) defining quotes in NetBill. In some cases, it may be required that quotes expire after a certain time—expired quotes may not lead to a contract. Moreover, we may need to denote when an active quote is close to expiring as a warning that the offer will soon be unavailable. To address these requirements, we may augment rules (2) and (3) with $\mathsf{fi}(quote(M, C, G) = \mathsf{true}, quote(M, C, G) = expiring, r_e)$ and $\mathsf{fi}(quote(M, C, G) = expiring, quote(M, C, G) = \mathsf{false}, r_f)$. The former fi fact expresses that $present\_quote(M, C, G, P)$, i.e., the event that leads to an immediate initiation of $quote(M, C, G) = \mathsf{true}$ (see rule (2)), leads to a future initiation of $quote(M, C, G) = expiring$, after a delay r_e, unless $quote(M, C, G) = \mathsf{true}$ is 'broken' in the meantime (based on rule (3), an $accept\_quote(C, M, G)$ event may 'break' $quote(M, C, G) = \mathsf{true}$). According to the latter fi fact, an initiation of the FVP $quote(M, C, G) = expiring$ leads to a future initiation of the FVP $quote(M, C, G) = \mathsf{false}$, after a delay of r_f time-points, provided that $quote(M, C, G) = expiring$ is not 'broken' in the meantime. In other words, the occurrence of $present\_quote(M, C, G, P)$ at T has the following effects, provided that no future initiation is cancelled: (a) FVP $quote(M, C, G) = \mathsf{true}$ holds in the interval $(T, T{+}r_e]$, (b) FVP $quote(M, C, G) = expiring$ holds in $(T{+}r_e, T{+}r_e{+}r_f]$, and (c) FVP $quote(M, C, G) = \mathsf{false}$ holds after $T{+}r_e{+}r_f$. □

A future initiation of $F = V'$ at time-point $T{+}R$, based, e.g., on $\mathsf{fi}(F = V, F = V', R)$, implies that $F = V$ is 'broken' at $T{+}R$. In RTEC$^{\rightarrow}$, it is also possible to express the future termination of an FVP $F = V$ without making any assertions about the future value of F, i.e., we become agnostic about the value of F in the future. This is a simpler case than future initiations, and thus not discussed here (see Appendix C for details on future terminations).

We may have an arbitrary set of $\mathsf{fi}(F = V, F = V', R)$ facts in an event description. However, it is not meaningful to state in the same event description both $\mathsf{fi}(F = V, F = V', R)$ and $\mathsf{fi}(F = V, F = V'', R')$, where $V \neq V' \neq V''$. Suppose, e.g., that $F = V$ is initiated at time-point T, $R < R'$ and $F = V$ is not cancelled between T and $T{+}R$. Then, $F = V'$ will be initiated at $T{+}R$, and thus $T{+}R$ will be a termination point of $F = V$. As a result, the future initiation of $F = V''$ at $T{+}R'$ will be cancelled. Therefore, in the case of $\mathsf{fi}(F = V, F = V', R)$ and $\mathsf{fi}(F = V, F = V'', R')$, only the future initiation with the shorter delay may take place.

In some cases, future initiations may be postponed.

Example 4 (Prolonged Quotes): In NetBill, we may allow for the possibility that a merchant M extends the interval during which a consumer C may accept a quote, and thus establish a contract between M and C. For instance, when $quote(M, C, G) = \mathsf{true}$, M may present another quote to consumer C, e.g., for a different price, with the aim of postponing the expiration of $quote(M, C, G) = \mathsf{true}$. To cater for this possibility, we may add $\mathsf{p}(quote(M, C, G) = \mathsf{true})$ in the event description. This way, a re-initiation of $quote(M, C, G) = \mathsf{true}$ will postpone its future termination. □

Example 5 (Streams of Events with Delayed Effects): Figure 2 visualises streams of events with delayed effects. In examples (a)(ii) and (b)(ii), we have a stream of three events that lead to immediate initiations of an FVP $F = V$. Suppose that these events take place at time-points T_1, T_2 and T_3, where $T_3{-}T_1{<}R$. According to $\mathsf{fi}(F = V, F = V', R)$, there may be future initiations of $F = V'$ at time-points $T_1{+}R$, $T_2{+}R$ and $T_3{+}R$. If $\mathsf{p}(F = V)$ is not in the event description, i.e., the future initiations of $F = V'$ may not be postponed (see example (a)(ii)), then we have a future initiation of $F = V'$ at $T_1{+}R$, since $F = V$ is not 'broken' between T_1 and $T_1{+}R$. As a result of this future initiation of $F = V'$, $F = V$ is 'broken' at $T_1{+}R$, and thus all subsequent future initiations of $F = V'$ are cancelled. In contrast, if $\mathsf{p}(F = V)$ is in the event description, i.e., the future initiations of $F = V'$ may be postponed (example (b)(ii)), the re-initiation of $F = V$ at T_2 moves the future initiation of $F = V'$ from $T_1{+}R$ to $T_2{+}R$, and the re-initiation of $F = V$ at T_3 moves the future initiation of $F = V'$ from $T_2{+}R$ to $T_3{+}R$. Two further illustrations of streams of events with delayed effects are presented in examples (a)(iii) and (b)(iii), and (a)(iv) and (b)(iv). Examples (a)(i) and (b)(i), and (a)(v) and (b)(v), concern windowing and will be discussed shortly. □

Using fi and p facts, RTEC$^{\rightarrow}$ can capture a wide range of delayed effects of events, such as regular payments from a costumer to a service provider (Farrell et al. 2005), timed changes in biological variables (Srinivasan et al. 2022), surviving obligations in legal contracts (Sharifi et al. 2020), temporally-restricted maritime activities such

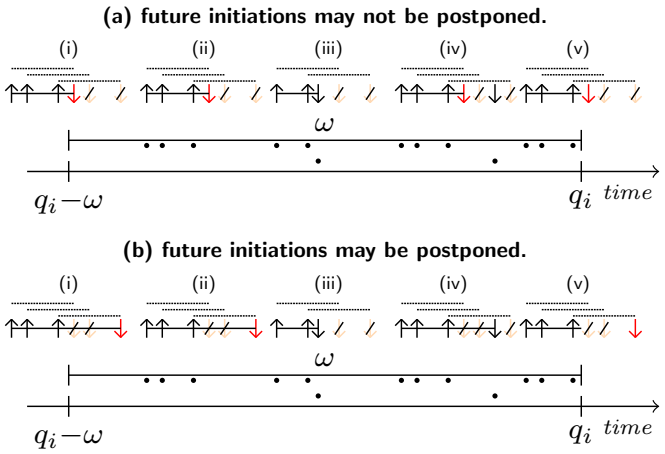

Fig. 2: Future initiations expressed by fi$(F=V, F=V', R)$ in a streaming setting. ω and q_i denote the window size and the i-th query time. The black points below ω represent the events of the stream. The events in the top (resp. bottom) row lead to immediate initiations (terminations) of $F=V$ (e.g., rules (2) and (3)). Arrows facing upwards indicate initiations of $F=V$. Red downward arrows express future terminations of $F=V$, which are the result of future initiations of $F=V'$, while black downward arrows indicate immediate terminations of $F=V$. Faded, crossed-out arrows denote cancelled future terminations of $F=V$, which are due to cancelled future initiations of $F=V'$. The dotted lines above the arrows express the delay R. Solid lines between arrows express the maximal intervals of $F=V$. In (a)(v) and (b)(v), the displayed intervals end at q_i to indicate that they are open, i.e., $F=V$ is not terminated within the current window.

as search and rescue operations (Pitsikalis et al. 2019), duration-fixed voting procedures (Pitt et al. 2006), and temporary suspensions in multi-agent e-commerce protocols (Artikis and Sergot 2010). In our experimental analysis (Section 6), we evaluated RTEC$^{\rightarrow}$ on several of these temporal specifications. In some cases, the delayed effects of events depend on numerical constraints. Consider, for example, an argumentation game (Artikis, Sergot, and Pitt 2007) where a proponent and an opponent present their arguments in turns, in a bounded number of rounds, say k rounds. RTEC$^{\rightarrow}$ is able to compute the termination of each turn—by setting up a future initiation for the turn of the next player. However, we cannot report the end of the game after k rounds, as this would require a condition in the future initiation of the next turn, stating that the value of a round-counting fluent must not exceed k. We chose to restrict attention to unconditional fi and p statements, i.e., fi and p statements without numerical constraints, to avoid compromising efficiency. Supporting numerical constraints, in the representation of delayed effects of events or elsewhere, is a direction for further research.

3.2 Semantics

fi facts introduce additional dependencies among FVPs. According to fi$(F=V, F=V', R)$, a future initiation of $F=V'$ depends on the initiation of $F=V$. We update the definition of a dependency graph as follows.

Definition 11 (Dependency Graph in RTEC$^{\rightarrow}$): The dependency graph of an event description in RTEC$^{\rightarrow}$ is a graph $G=(\mathcal{V}, \mathcal{E}_i \cup \mathcal{E}_f)$ with directed edges, where:

- $\mathcal{V}$ contains one vertex $v_{F=V}$ for each FVP $F=V$.
- $\mathcal{E}_i$ contains 'i-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.

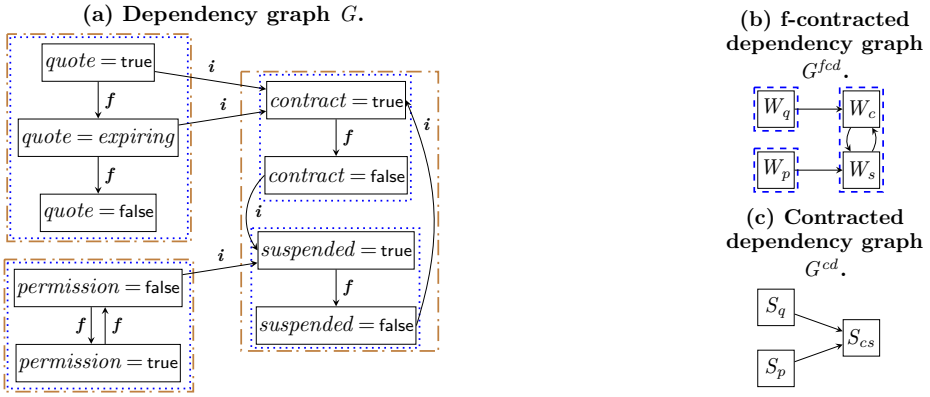

Fig. 3: (a) The dependency graph G, (b) the f-contracted dependency graph G^{fcd} and (c) the contracted dependency graph G^{cd} of NetBill. For simplicity, we omitted the arguments of fluents, while a vertex v_j is displayed as j. In diagram (a), the blue dotted rectangles denote the weakly connected components of the i-reduced dependency graph G^{ird} of G. The brown dotted-dashed rectangles denote the cd-components of G. In diagram (b), the blue dashed rectangles denote the strongly connected components of G^{fcd}. The vertices of W_q, W_p, W_c and W_s are the result of contracting the weakly connected components of G^{ird}, concerning resp. the FVPs of *quote*, *permission*, *contract* and *suspended*. In diagram (c), the vertices of S_q, S_p and S_{cs} are produced by contracting the strongly connected components of G^{fcd}, concerning W_q, W_p, and W_c and W_s.

- $\mathcal{E}_f$ contains 'f-edge' $(v_{F=V}, v_{F=V'})$ iff there is a fact $\mathsf{fi}(F=V, F=V', R)$, and for every $\mathsf{fi}(F=V, F=V'', R')$ fact, where $V' \neq V''$, if any, it holds that $R < R'$. ■

In other words, the edges of RTEC dependency graphs (see Definition 8), i.e., the edges expressing immediate initiations/terminations, are now called 'i-edges', in order to distinguish them from the edges expressing future initiations/terminations, i.e., 'f-edges'.

Example 6 (Dependency Graph): Figure 3(a) presents a fragment of the dependency graph of an event description for NetBill. i-edges denote FVP dependencies based on rules with head initiatedAt or terminatedAt. For example, the i-edges pointing to the vertex of $contract(M, C, G) = \mathsf{true}$ express that $contract(M, C, G) = \mathsf{true}$ may be initiated, i.e., a contract between a merchant M and a consumer C about goods G may come into effect, when $quote(M, C, G) = \mathsf{true}$ or $quote(M, C, G) = expiring$, i.e., the quote has not yet expired. (Recall that events, such as $accept\_quote(C, M, G)$ that may initiate $contract(M, C, G)$, are not included in dependency graphs.) f-edges model dependencies based on fi facts. f-edge $(v_{quote(M,C,G)=\mathsf{true}}, v_{quote(M,C,G)=expiring})$, e.g., is due to $\mathsf{fi}(quote(M, C, G) = \mathsf{true}, quote(M, C, G) = expiring, r_e)$ (Example 3). □

A dependency graph in RTEC$^{\rightarrow}$ may include an f-edge, such as $(v_{F=V}, v_{F=V'})$, according to which, following Definition 9, $F=V'$ has a higher level than $F=V$. However, $F=V$ and $F=V'$ depend on each other, i.e., the initiations of $F=V'$ terminate $F=V$, and vice versa. Therefore, $F=V$ and $F=V'$ should remain in the same level. Moreover, a dependency graph in RTEC$^{\rightarrow}$ may include cycles (see, e.g., Figure 3(a)). To address these issues, we have to extend the definition of 'FVP level' (Definition 9), which is necessary for stratifying RTEC$^{\rightarrow}$ programs, and guides the reasoning algorithms. In what follows, we present the steps that lead to the new definition of FVP level. Subsequently, we present the semantics of RTEC$^{\rightarrow}$, and in the following section, we present the reasoning algorithms.

In order to define the level of FVPs in an RTEC$^{\rightarrow}$ event description, we first transform its dependency graph G into the ***f-contracted*** dependency graph G^{fcd}, where the vertices connected by f-edges have been contracted into a single vertex. To do this, we construct the ***i-reduced*** dependency graph G^{ird} of G, which is generated by removing all i-edges from G, and contract the vertices that are connected in G^{ird}.

Definition 12 (f-Contracted Dependency Graph): Given an RTEC$^{\rightarrow}$ event description with dependency graph $G=(\mathcal{V}, \mathcal{E}_i \cup \mathcal{E}_f)$, the i-reduced dependency graph $G^{ird}=(\mathcal{V}, \mathcal{E}_f)$ of G, and the weakly connected components (WCC)s $W_1, W_2, \ldots, W_n$ of G^{ird}, we define the f-contracted dependency graph $G^{fcd}=(\mathcal{V}^{fcd}, \mathcal{E}^{fcd})$ of G as follows:

- $\mathcal{V}^{fcd} = \bigcup_{1 \leq j \leq n} \{v_{W_j}\}$.
- $(v_{W_j}, v_{W_k}) \in \mathcal{E}^{fcd}$ iff $\exists v_j, v_k \in \mathcal{V}$: $v_j \in W_j$, $v_k \in W_k$ and $(v_j, v_k) \in \mathcal{E}_i$. ■

Example 7 (f-Contracted Dependency Graph): Figure 3(b) depicts the f-contracted dependency graph of NetBill. The i-reduced dependency graph is the same as the graph of Figure 3(a) without the i-edges. The vertices of FVPs of *quote* and *contract*, e.g., constitute WCCs of the i-reduced dependency graph, denoted by W_q and W_c, and thus have been contracted into vertices v_{W_q} and v_{W_c}, respectively. Moreover, there is an edge (v_{W_q}, v_{W_c}) in the f-contracted dependency graph because the dependency graph includes i-edges from the vertices of *quote* to a vertex of *contract*. □

An f-contracted dependency graph may include cycles (see, e.g., Figure 3(b)). To define the FVP level in the presence of cycles, we contract G^{fcd} based on its strongly connected components (SCC)s (after (Mantenoglou, Pitsikalis, et al. 2022)), generating the ***contracted*** dependency graph G^{cd} of G.

Definition 13 (Contracted Dependency Graph): Given the f-contracted dependency graph $G^{fcd}=(\mathcal{V}^{fcd}, \mathcal{E}^{fcd})$ of a dependency graph G in RTEC$^{\rightarrow}$, and the maximal SCCs $S_1, S_2, \ldots, S_n$ of G^{fcd}, the contracted dependency graph $G^{cd}=(\mathcal{V}^{cd}, \mathcal{E}^{cd})$ of G is defined as follows:

(1) $\mathcal{V}^{cd} = \bigcup_{1 \leq j \leq n} \{v_{S_j}\}$.

(2) $(v_{S_j}, v_{S_k}) \in \mathcal{E}^{cd}$ iff $\exists v_j, v_k \in \mathcal{V}^{fcd}$: $v_j \in S_j$, $v_k \in S_k$, where $S_j \neq S_k$, and $(v_j, v_k) \in \mathcal{E}^{fcd}$. ■

By construction, G^{cd} is acyclic.

Example 8 (Contracted Dependency Graph): Figure 3(c) presents the contracted dependency graph of NetBill. Vertices v_{W_c} and v_{W_s} of the f-contracted dependency graph constitute a SCC, and thus have been contracted into a single vertex $v_{S_{cs}}$, in order to produce the contracted dependency graph. The edges $(v_{S_q}, v_{S_{cs}})$ and $(v_{S_p}, v_{S_{cs}})$ in the contracted dependency graph are due to the edges (v_{W_q}, v_{W_c}) and (v_{W_p}, v_{W_s}) in the f-contracted graph. □

We say that the vertices of FVPs that are mapped onto the same vertex in G^{cd} comprise a *cd-component* of G. We call the union of the cd-components of G its *cd-partition*.

Definition 14 (cd-partition and cd-component): Consider a dependency graph G in RTEC$^{\rightarrow}$, the i-reduced dependency graph G^{ird} of G, and the f-contracted dependency graph G^{fcd} of G, where $S_1, \ldots, S_n$ are the maximal SCCs of G^{fcd}. $G = G_{S_1} \cup \cdots \cup G_{S_n}$ is the cd-partition of G iff, for each $F=V$ such that $v_{F=V} \in G_{S_i}$, $v_{F=V}$ is included in a WCC W of G^{ird} such that v_W is included in SCC S_i of G^{fcd}. We call G_{S_i} a cd-component of G. ■

According to Definition 14, there is a one-to-one mapping between the cd-components of G and the vertices of G^{cd}.

Example 9 (cd-components): In the dependency graph of NetBill, we have three cd-components G_{S_q}, G_{S_p} and $G_{S_{cs}}$, containing, respectively, the vertices of *quote*, the vertices of *permission*, and the vertices of *contract* and *suspended* (see Figure 3(a)). □

Definition 15 (FVP Level in RTEC$^{\rightarrow}$): Given a dependency graph G in RTEC$^{\rightarrow}$, its cd-components $G_{S_1}, \ldots, G_{S_n}$ and the contracted dependency graph G^{cd} of G, the level of an FVP $F=V$, such that $v_{F=V} \in G_{S_i}$, is defined as the level of the vertex v_{S_i} in G^{cd}. ■

Example 10 (FVP Level): In the contracted dependency graph of NetBill (Figure 3(c)), the vertices v_{S_q} and v_{S_p} have level 1, while $v_{S_{sc}}$ has level 2. Therefore, the FVPs of *quote* and *permission* have level 1, and the FVPs of *contract* and *suspended* have level 2. □

The semantics of RTEC$^{\rightarrow}$ remains a locally stratified logic program. The definition of FVP level allows us to arrange the initiatedAt and the terminatedAt predicates of FVPs into strata in ascending FVP level order. For FVPs of the same cd-component, and thus the same level, a stratification may be constructed by introducing an additional stratum for each time-point T of the window, in ascending temporal order.

Proposition 2 (Semantics of RTEC$^{\rightarrow}$): An event description in RTEC$^{\rightarrow}$ is a locally stratified logic program. ▲

4 Reasoning

4.1 Naive Event Calculus

One way to handle events with delayed effects would be to employ the following rules:

$$\begin{aligned} &\mathsf{initiatedAt}(F=V', T{+}R) \leftarrow \\ &\quad \mathsf{fi}(F=V, F=V', R),\ \mathsf{initiatedAt}(F=V, T), \\ &\quad \mathsf{not\ cancelled}(F=V, T, T{+}R). \end{aligned} \tag{α}$$

$$\begin{aligned} &\mathsf{cancelled}(F=V, T, T{+}R) \leftarrow \\ &\quad \mathsf{brokenBetween}(F=V, T, T{+}R). \end{aligned} \tag{β}$$

$$\begin{aligned} &\mathsf{cancelled}(F=V, T, T{+}R) \leftarrow \\ &\quad \mathsf{p}(F=V),\ \mathsf{initiatedBetween}(F=V, T, T{+}R). \end{aligned} \tag{γ}$$

According to rule (α), $F=V'$ will be initiated at $T{+}R$ if an initiation of $F=V$ leads to an initiation of $F=V'$ with delay R, indicated by $\mathsf{fi}(F=V, F=V', R)$, $F=V$ is indeed initiated at T, and the future initiation is not 'cancelled' between T and $T{+}R$. Rule (β) expresses that the future initiation of $F=V'$ will be cancelled if $F=V$ is 'broken' between T and $T{+}R$, i.e., $F=V$ is terminated or $F=V''$, where $V'' \neq V$, is initiated. Rule (γ) concerns future initiations that may be postponed, as indicated by $\mathsf{p}(F=V)$, and expresses that the future initiation of $F=V'$ will be cancelled if $F=V$ is initiated between T and $T{+}R$. In other words, the future initiation of $F=V'$ may be postponed by re-initiating $F=V$.

Unfortunately, rules (α)–(γ) are not optimised for reasoning over data streams. The evaluation of these rules may lead to the computation of initiation points that have already been computed. See, e.g., Figure 2(a)(ii), where we have three initiations of $F=V$ at time-points T_1, T_2 and T_3, and a future initiation of $F=V'$ at $T_1{+}R$. When computing the effects of the initiations of $F=V$ at T_2 and T_3, we would have to prove twice more the future initiation of $F=V'$ at $T_1{+}R$, only to cancel the future initiations of $F=V'$ at $T_2{+}R$ and $T_3{+}R$. Furthermore, rules (α)–(γ) are not designed to operate in the presence of windowing, which is quintessential for streaming applications. See, e.g., Figure 2(a)(i), where the events initiating $F=V$ take place before the start of the current window, while the future initiations are said to take place after the start of the window.

For these reasons, rules (α)–(γ) are not part of RTEC$^{\rightarrow}$. To address the aforementioned issues, we have developed a caching algorithm that avoids unnecessary re-computations when reasoning over events with delayed effects, and an algorithm identifying the minimal information that needs to be transferred between sliding windows in order to guarantee correctness. These algorithms are presented in Section 4.3. Below, in Section 4.2, we present a compile-time process computing the optimal FVP processing order in the presence of an arbitrary set of fi relations.

4.2 Off-Line Reasoning

Based on Definition 15, RTEC$^{\rightarrow}$ can process the FVPs of an event description in a bottom-up fashion, according to their level, caching the FVP intervals derived at each level. This way, when processing an FVP $F=V$ of level m,

the intervals of the FVPs of levels lower than m that participate in the definition of $F = V$, if any, may be retrieved from the cache without the need for re-computation. This processing order, however, is not applicable to FVPs of the same level, e.g., FVPs with fi relations.

Example 11 (Processing Quotes): Consider the cd-component G_{S_q} of the dependency graph of NetBill (Figure 3(a), top-left). Suppose that $quote(M, C, G) = \text{true}$ is initiated at time-points T_1 and T_2, and that future initiations are not cancelled. Based on the fi facts (see Example 3), $quote(M, C, G) = expiring$ is initiated at $T_1{+}r_e$ and $T_2{+}r_e$, and $quote(M, C, G) = \text{false}$ is initiated at $T_1{+}r_e{+}r_f$ and $T_2{+}r_e{+}r_f$. When processing the FVPs in G_{S_q}, starting with FVP $quote(M, C, G) = \text{false}$ leads to redundant computations. In order to compute the future initiations of $quote(M, C, G) = \text{false}$ that fall within the window $(q_i{-}\omega, q_i]$, we need to evaluate whether $quote(M, C, G) = expiring$ is initiated at each time-point in $(q_i{-}\omega, q_i{-}r_f]$. In turn, in order to compute the future initiations of $quote(M, C, G) = expiring$ in $(q_i{-}\omega, q_i{-}r_f]$, we need to evaluate the initiations of $quote(M, C, G) = \text{true}$ at each time-point in $(q_i{-}\omega, q_i{-}r_f{-}r_e]$.

We can avoid redundant computations at time-points where we may not have future initiations by processing the FVPs of G_{S_q} in an order induced by the f-edges of G_{S_q}. This way, we first compute the initiation points T_1 and T_2 of $quote(M, C, G) = \text{true}$ by evaluating $\text{initiatedAt}(quote(M, C, G) = \text{true}, T)$ at each time-point of the window. Next, given these initiation points, we can compute the initiations of $quote(M, C, G) = expiring$ at $T_1{+}r_e$ and $T_2{+}r_e$, without considering the remaining time-points of the window. Similarly, given the initiation points $T_1{+}r_e$ and $T_2{+}r_e$ of $quote(M, C, G) = expiring$, we can compute the initiations of $quote(M, C, G) = \text{false}$ at $T_1{+}r_e{+}r_f$ and $T_2{+}r_e{+}r_f$, without any redundant evaluations. □

In order to avoid unnecessary computations, RTEC$^{\rightarrow}$ evaluates and caches the initiations of the FVPs in a cd-component in an order induced by the f-edges.

Definition 16 (FVP Order Relation): Consider the dependency graph G of an event description in RTEC$^{\rightarrow}$ and the cd-components $G_{S_1}, \ldots, G_{S_n}$ of G. For a cd-component G_{S_i}, we define the FVP order relation $\prec_{G_{S_i}}$ on the FVPs whose vertices are in G_{S_i} as follows:

- If G_{S_i} is acyclic, then $F = V \prec_{G_{S_i}} F' = V'$ iff there is a path in G_{S_i} that starts from $v_{F=V}$ and ends at $v_{F'=V'}$.
- If G_{S_i} contains a cycle, then relation $\prec_{G_{S_i}}$ is empty. ■

In cd-component G_{S_q}, e.g., we have $quote = \text{true} \prec_{G_{S_q}} quote = expiring \prec_{G_{S_q}} quote = \text{false}$.

The FVP order relation $\prec_{G_{S_i}}$ is a strict partial order, meaning that some FVPs may not be comparable with $\prec_{G_{S_i}}$, even in an acyclic cd-component.

Example 12 (Partial FVP Order Relation): Consider an event description containing $\text{fi}(F = V_1, F = V_3, R_1)$ and $\text{fi}(F = V_2, F = V_3, R_2)$. The dependency graph has a cd-component G_{S_i}, containing the vertices of $F = V_1$, $F = V_2$ and $F = V_3$. Based on f-edges $(v_{F=V_1}, v_{F=V_3})$ and $(v_{F=V_2}, v_{F=V_3})$, we have $F = V_1 \prec_{G_{S_i}} F = V_3$ and $F = V_2 \prec_{G_{S_i}} F = V_3$. However, there is no directed path connecting vertices $v_{F=V_1}$ and $v_{F=V_2}$, and thus $F = V_1$ and $F = V_2$ are not ordered with relation $\prec_{G_{S_i}}$. □

Based on $\prec_{G_{S_i}}$, we define a total processing order of FVPs as follows:

Definition 17 (Processing Order of FVPs): Given the FVP order relation $\prec_{G_{S_i}}$ of a cd-component G_{S_i} of a dependency graph in RTEC$^{\rightarrow}$, a processing order of FVPs is any strict total order $\prec^*_{G_{S_i}}$ such that, for each pair of FVPs $F = V$ and $F' = V'$, if we have $F = V \prec_{G_{S_i}} F' = V'$, then $F = V \prec^*_{G_{S_i}} F' = V'$, i.e., $\prec^*_{G_{S_i}}$ is a linear extension of $\prec_{G_{S_i}}$. ■

Example 13 (Processing order of FVPs): For the event description of Example 12, $\prec^*_{G_{S_i}}$ is consistent with $\prec_{G_{S_i}}$, i.e., we have $F = V_1 \prec^*_{G_{S_i}} F = V_3$ and $F = V_2 \prec^*_{G_{S_i}} F = V_3$. Moreover, since FVPs $F = V_1$ and $F = V_2$ are not comparable with $\prec_{G_{S_i}}$, $\prec^*_{G_{S_i}}$ extends $\prec_{G_{S_i}}$ with $F = V_1 \prec^*_{G_{S_i}} F = V_2$ or $F = V_2 \prec^*_{G_{S_i}} F = V_1$. □

Algorithm 1 processCDComponent

Input: cd-component G_{S_i}, cached intervals I.
Output: I, including the intervals of the FVPs in G_{S_i}.

1: **if** G_{S_i} does not contain a cycle **then**
2: **for each** $v_{F=V}$ **in** G_{S_i} **do** $IP[F=V] \leftarrow [\,]$
3: **for each** $v_{F=V}$ **in** processingOrder(G_{S_i}) **do**
4: $IP[F=V]$.add(RTEC.evalIP($F=V, I$))
5: $TP[F=V] \leftarrow$RTEC.evalTP($F=V, I$)
6: **if** fi($F=V, F=V', R$) **then**
7: $FtIP[F=V'] \leftarrow$evalFI($IP[F=V]$, $TP[F=V]$, V', R)
8: $TP[F=V]$.add($FtIP[F=V']$)
9: $IP[F=V']$.add($FtIP[F=V']$)
10: $I[F=V] \leftarrow$RTEC.mi($IP[F=V]$, $TP[F=V]$)
11: **else if** G_{S_i} does not contain an f-edge **then**
12: I.updateIntervals(RTEC$_\circ(G_{S_i}, I)$)
13: **else** I.updateIntervals(cyclicFI(G_{S_i}, I))
14: **return** I

The processing order of FVPs is derived at compile-time, in a process transparent to the event description developer. During run-time operations, RTEC$^{\rightarrow}$ utilises this processing order to guide reasoning over FVPs of the same level.

4.3 Run-Time Reasoning

RTEC$^{\rightarrow}$ processes the cd-components of a dependency graph level-by-level, and, for each cd-component G_{S_i}, RTEC$^{\rightarrow}$ computes and caches the future initiations and the maximal intervals of the FVPs whose vertices are in G_{S_i}, by following processing order $\prec^*_{G_{S_i}}$. For the dependency graph of NetBill (Figure 3(a)), e.g., RTEC$^{\rightarrow}$ processes G_{S_q} or G_{S_p} first, and $G_{S_{cs}}$ last, because the FVPs in G_{S_q} and G_{S_p} have level 1, and the FVPs in $G_{S_{cs}}$ have level 2. Moreover, for cd-component G_{S_q}, RTEC$^{\rightarrow}$ follows $\prec^*_{G_{S_q}}$, computing and caching the future initiations and the intervals of the FVPs in the following order: $quote(M, C, G) = \text{true}$, $quote(M, C, G) = expiring$, and, finally, $quote(M, C, G) = \text{false}$. This way, RTEC$^{\rightarrow}$ does not perform any redundant computation.

RTEC$^{\rightarrow}$ also identifies the attempts of initiating an FVP at some future time-point that need to be transferred between sliding windows, in order to guarantee correctness. Below, we present the key reasoning algorithms of RTEC$^{\rightarrow}$; the remaining ones are in Appendix A.

Algorithm 1 presents the steps for deriving the maximal intervals of the FVPs in a cd-component G_{S_i}. I contains the intervals of FVPs in previously processed cd-components. We start with the case where there is no cycle in G_{S_i} (lines 1–10). RTEC$^{\rightarrow}$ processes the FVPs in G_{S_i} following processing order $\prec^*_{G_{S_i}}$ (line 3), and, for each FVP $F=V$, RTEC$^{\rightarrow}$ employs RTEC (see Section 2) to evaluate the initiatedAt and terminatedAt rules of $F=V$ (lines 4–5). The derived initiation points of $F=V$ are added in list $IP[F=V]$, while list $TP[F=V]$ contains the derived termination points. Next, based on fi($F=V, F=V', R$), RTEC$^{\rightarrow}$ evaluates the future initiations of $F=V'$, as they constitute terminations of $F=V$ (line 7). The details of this process will be presented shortly. RTEC$^{\rightarrow}$ stores the time-points of these future initiations in the auxiliary list $FtIP[F=V']$, and adds the items of $FtIP[F=V']$ to $TP[F=V]$. Moreover, RTEC$^{\rightarrow}$ adds the items of $FtIP[F=V']$ to $IP[F=V']$; this way, the future initiations of $F=V'$ will be available to RTEC$^{\rightarrow}$ when processing $F=V'$. At this point, lists $IP[F=V]$ and $TP[F=V]$ contain all initiation and termination points of $F=V$, and no other time-points (see Section 5 for the correctness proof).

Algorithm 2 evalFI

Input: $IP[F=V]$, $TP[F=V]$, V', R.
Global: q_i, ω.
Output: The future initiations of $F=V'$.
1: $FtIP[F=V'] \leftarrow [\,]$
2: $T_{att} \leftarrow \mathsf{findAttempt}(F=V, R)$
3: **if** $T_{att} \neq null$ **and** not $\mathsf{cancelled}(q_i{-}\omega, T_{att}, IP[F=V], TP[F=V])$ **then**
4: $\quad FtIP[F=V'].\mathsf{add}(T_{att})$
5: **for each** T **in** $IP[F=V]$ **do**
6: $\quad$ **if** $T{+}R \leq q_i$ **and** not $\mathsf{cancelled}(T, T{+}R, IP[F=V], TP[F=V] \cup FtIP[F=V'])$ **then**
7: $\quad\quad FtIP[F=V'].\mathsf{add}(T{+}R)$
8: $\mathsf{computeAttempts}(F=V, R)$
9: **return** $FtIP[F=V']$

RTEC$^{\rightarrow}$ is then able to construct the maximal intervals of $F=V$ (see RTEC.mi in line 10), by pairing initiation and termination points (see Section 2).

A cd-component G_{S_i} may contain cycles (see, e.g., G_{S_p} and $G_{S_{cs}}$ in Figure 3(a)). If G_{S_i} contains cycles without f-edges, then RTEC$^{\rightarrow}$ employs RTEC$_{\circ}$, an extension of RTEC handling cyclic dependency graphs (Mantenoglou, Pitsikalis, et al. 2022), to compute the intervals of FVPs in G_{S_i} (see line 12 of Algorithm 1 and Appendix A.3). For cycles with at least one f-edge (line 13), RTEC$^{\rightarrow}$ employs an extended version of the technique presented in (Mantenoglou, Pitsikalis, et al. 2022), handling future initiations in the presence of cycles.

In the following, we describe the steps of RTEC$^{\rightarrow}$ for computing the future initiations of $F=V'$, that lead to terminations of $F=V$. Due to windowing, it is possible for an initiation of $F=V$ to take place at a time-point T of the current window, while the corresponding future initiation of $F=V'$ at $T{+}R$ falls outside the window. In such cases, RTEC$^{\rightarrow}$ generates and caches an attempt event *att* that may mark a future initiation $F=V'$ at $T{+}R$. At each query time, RTEC$^{\rightarrow}$ evaluates the future initiations of $F=V'$ based on the cached attempt events, and the events that fall within the window. Algorithm 2 presents the reasoning process. First, RTEC$^{\rightarrow}$ processes the attempt events cached at the previous query time, in order to determine which one of them, if any, may mark a future initiation of $F=V'$ at the current query time (line 2 and Appendix A.1). Second, RTEC$^{\rightarrow}$ computes the future initiations of $F=V'$, based on the selected attempt event, if any, and the events that took place inside the window (lines 3–7). Third, RTEC$^{\rightarrow}$ computes and caches attempt events that may mark future initiations of $F=V'$ at a future query time (line 8 and Appendix A.2). In what follows, we present each of these three steps.

Processing attempt events from the previous query time. RTEC$^{\rightarrow}$ decides which one of the attempt events that were computed and cached at the previous query time q_{i-1}, if any, may mark a future initiation of $F=V'$ at q_i. First, we make sure that the future initiation of $F=V'$ was not cancelled before the current window by checking whether $F=V$ holds at the start of the window. If it holds, then we keep an attempt event such that the corresponding initiation of $F=V$ takes place before $q_i{-}\omega$. If $\mathrm{p}(F=V)$ is not in the event description, then we keep the earliest such event. Otherwise, we choose the most recent one. Note that it is not possible to determine at q_{i-1} which attempt event will be used at q_i, because we make no assumptions about the size of the window at q_i.

Example 14 (Processing attempt events from the previous query time): Consider Figures 2(a)(i) and (b)(i). For each initiation of $F=V$ at T, that took place before the current window, with $T{+}R$ falling within the window, we have a cached attempt event at $T{+}R$. These events are displayed by red and faded arrows. The red arrow denotes the attempt event *att* that is kept, while the faded arrows denote the attempt events that are discarded.

In (a)(i), future initiations of $F = V'$ may not be postponed, and thus *att* takes place R time-points after the first initiation of $F = V$. In (b)(i), *att* takes place R time-points after the last initiation of $F = V$, since future initiations of $F = V'$ may be postponed. □

Computing future initiations. After determining which cached attempt event *att*, if any, may mark a future initiation of $F = V'$, RTEC$^{\rightarrow}$ computes the future initiations of $F = V'$ that fall within the current window. First, RTEC$^{\rightarrow}$ evaluates whether *att* marks a future initiation of $F = V'$ by checking whether the future initiation of $F = V'$ is cancelled between the start of the window and the time-stamp T_{att} of *att* (see lines 3–4 of Algorithm 2). Second, for each initiation of $F = V$ at T, RTEC$^{\rightarrow}$ computes a future initiation of $F = V'$ at T+R, provided that T+R falls inside the window and the future initiation of $F = V'$ is not cancelled between T and T+R (lines 5–7). $IP[F = V]$, like all lists in RTEC$^{\rightarrow}$, is temporally sorted; thus, RTEC$^{\rightarrow}$ examines the initiations of $F = V$ in ascending temporal order. Lines 5–7 are an efficient implementation of rules (α)–(γ). In order to identify an initiation of $F = V$ that may lead to a future initiation of $F = V'$, RTEC$^{\rightarrow}$ examines the list of cached initiation points of $F = V$, without requiring any rule computations (contrast line 5 of Algorithm 2 with the second condition of rule (α)). Moreover, in order to evaluate whether a future initiation of $F = V'$ is cancelled between T and T+R, RTEC$^{\rightarrow}$ checks if there is a cached initiation or termination of $F = V$ between T and T+R that cancels the future initiation of $F = V'$. Instead, rules (β)–(γ) evaluate initiations and terminations of $F = V$ that may have been previously computed.

Example 15 (Future initiations within window): Consider example (a)(ii) of Figure 2. RTEC$^{\rightarrow}$ computes and caches the initiations of $F = V$ at T_1, T_2 and T_3. RTEC$^{\rightarrow}$ then determines whether we have future initiations of $F = V'$ at T_1+R, T_2+R and T_3+R (see lines 5–7 of Algorithm 2). RTEC$^{\rightarrow}$ starts with the initiation point of $F = V$ at T_1, and computes a future initiation of $F = V'$ at T_1+R, because, according to the cached points, the future initiation of $F = V'$ is not cancelled between T_1 and T_1+R. As a result, T_1+R is cached in list $FtIP[F = V']$. Next, RTEC$^{\rightarrow}$ determines that the future initiations of $F = V'$ at T_2+R and T_3+R are cancelled, by retrieving T_1+R from $FtIP[F = V']$.

In example (a)(iii), $F = V$ is initiated twice, at time-points T_1 and T_2, and terminated once, at T_3. For the initiation of $F = V$ at T_1 (resp. T_2), RTEC$^{\rightarrow}$ determines that the future initiation of $F = V'$ at T_1+R (T_2+R) is cancelled by verifying that the cached termination point T_3 of $F = V$ is between T_1 (T_2) and T_1+R (T_2+R) (see lines 5–7 of Algorithm 2). Example (a)(iv) is similar to (a)(iii), with the exception that the termination point of $F = V$ takes place between the first and the last future initiation of $F = V'$. In this case, RTEC$^{\rightarrow}$ detects that only the first future initiation of $F = V'$ takes place, as in (a)(ii). In examples (b)(ii)–(iv), the future initiations of $F = V'$ may be postponed by a re-initiation of $F = V$. In (b)(ii), e.g., RTEC$^{\rightarrow}$ determines that the future initiations of $F = V'$ at T_1+R and T_2+R are cancelled, because they are postponed by the cached initiation T_3 of $F = V$ (lines 5–7 of Algorithm 2). Thus, we have a future initiation of $F = V'$ at T_3+R. □

Computing attempt events for the next query time. In the last step at q_i, RTEC$^{\rightarrow}$ derives the attempt events that may lead to future initiations of $F = V'$ at the next query time q_{i+1}. Recall that an attempt event takes place R time-points after the initiation of $F = V$. Moreover, we make no assumptions about the size of the window at q_{i+1}. If future initiations of $F = V'$ may be postponed, then every attempt event may mark an initiation of $F = V'$ at q_{i+1}, and is therefore cached. Otherwise, if future initiations of $F = V'$ may not be postponed, then we cache only the attempt events that take place R time-points after the starting point of a maximal interval of $F = V$.

Example 16 (Computing attempt events for the next query time): Consider examples (a)(v) and (b)(v) of Figure 2. In the case of (a)(v), only the attempt event indicated by the red arrow will be cached, while in (b)(v) the attempt events denoted by the red arrow and the faded ones will be cached. If we assume non-overlapping windows in the case of (b)(v), i.e., the next window starting at q_i, then the attempt events of the faded arrows will be discarded at q_{i+1}. However, if the next window starts between two initiations denoted by upward arrows, we will discard all attempt events corresponding to initiations that fall within the next window, because these

initiations may be invalidated in the presence of the events that arrive in $(q_i, q_{i+1}]$ and have a time-stamp in $(q_{i+1}-\omega, q_i]$. □

5 Correctness and Complexity

RTEC$^{\rightarrow}$ guarantees correct FVP interval computation provided that the window size is sufficient to tolerate the lags, if any, in the arrival of events. In other words, for each event E with time-stamp T, there is a query time q_i, such that E has arrived by q_i and $T \in (q_i-\omega, q_i]$. Please recall that the lags in the arrival of events are orthogonal to the delayed effects of events. Moreover, we make no assumptions concerning the extent of the delay in the effects of an event.

Proposition 3 (Correctness of RTEC$^{\rightarrow}$): RTEC$^{\rightarrow}$ computes all maximal intervals of the FVPs of an event description, and no other interval. ▲

Proof Sketch. We present a sketch of the proof, focusing on acyclic dependency graphs. The complete proof, including cyclic graphs, may be found in Appendix B. Assume an event description with $\mathrm{fi}(F=V, F=V', R)$. The proof consists of three steps. First, we prove that RTEC$^{\rightarrow}$ transfers the attempt events between windows that are required for correct FVP interval computation, and no other attempt event. Second, we show that RTEC$^{\rightarrow}$ computes all future initiations of $F=V'$, and no other future initiation. Third, at the time of computing the maximal intervals of an FVP, the cache contains all necessary initiations and terminations of the FVP, and no other initiation/termination.

We present the proof of the second step. At the start of future initiation computation, the cache contains all initiations and immediate terminations of $F=V$, and no other initiation or immediate termination of $F=V$. Suppose that $T_1, \ldots, T_k$ are the temporally sorted initiations of $F=V$. We show that RTEC$^{\rightarrow}$ computes a future initiation of $F=V'$ at T_i+R, where $1 \leq i \leq k$, iff the future initiation of $F=V'$ is not cancelled between T_i and T_i+R. For the base case, according to lines 5–7 of Algorithm 2, RTEC$^{\rightarrow}$ computes a future initiation of $F=V'$ at T_1+R iff there is no cancellation point between T_1 and T_1+R. Note that there are no initiations of $F=V$ before T_1, and thus no future initiations of $F=V'$ before T_1+R. In other words, the cache has all cancellation points of the future initiation of $F=V'$ between T_1 and T_1+R. Next, assume that RTEC$^{\rightarrow}$ has evaluated correctly the future initiations of $F=V'$ up to T_{n-1}+R, where $1 < n \leq k$. Since RTEC$^{\rightarrow}$ caches the future initiations that it derives (line 7 of Algorithm 2), the cache contains all cancellation points of the future initiations of $F=V'$ up to T_{n-1}+R. As a result, RTEC$^{\rightarrow}$ computes a future initiation of $F=V'$ at T_n+R iff there is no cancellation point between T_n and T_n+R.

The above proof may be generalised to a cd-component with an arbitrary set of f-edges. Such a cd-component may be built by combining the following two structures. The first structure is a 'chain' $(v_{F=V_1}, v_{F=V_2})$, $(v_{F=V_2}, v_{F=V_3})$, …, $(v_{F=V_n}, v_{F=V_{n+1}})$ of n f-edges. The second structure consists of a vertex $v_{F=V_m}$ with n incoming f-edges $(v_{F=V_1}, v_{F=V_m})$, …, $(v_{F=V_n}, v_{F=V_m})$ (see Example 12). In both cases, at the time of evaluating the future initiation of an FVP, the cache contains all the cancellation points that are necessary for correct computation. In the former case, based on Definition 17, RTEC$^{\rightarrow}$ processes the FVPs in the following order: $F=V_1, F=V_2, \ldots, F=V_{n+1}$. As a result, when processing $F=V_k$, where $1 < k \leq n+1$, the cache of RTEC$^{\rightarrow}$ contains all initiations and immediate terminations of $F=V_{k-1}$. At this point, the future initiations of $F=V_{k+j}$, where $1 \leq j \leq n-k+1$, are not required, because they do not terminate $F=V_{k-1}$, and thus do not cancel the future initiations of $F=V_k$. In general, a future initiation of $F=V_{k+j}$, based on f-edge $(v_{F=V_{k+j-1}}, v_{F=V_{k+j}})$, terminates $F=V_{k+j-1}$, and no other FVP. In the latter case, RTEC$^{\rightarrow}$ processes FVPs $F=V_1, \ldots, F=V_n$ before $F=V_m$. As a result, when processing $F=V_m$, the cache of RTEC$^{\rightarrow}$ contains all the initiations and immediate terminations of FVPs $F=V_1, \ldots, F=V_n$. Therefore, in both cases, the cache of RTEC$^{\rightarrow}$ contains the necessary information for correct future initiation computation. ◊

Recall that RTEC$^{\rightarrow}$ processes the cd-components of a dependency graph level-by-level. The proposition below presents the complexity of processing a cd-component.

Proposition 4 (Complexity of RTEC$^{\rightarrow}$): The worst-case cost of evaluating the future initiations of the FVPs in a cd-component is $O(n_v(\omega{-}R)log(\omega))$, where n_v is the number of possible values of the FVPs, ω is the window size and R is the delay of a future initiation. ▲

Proof. See Appendix B. ◊

Proposition 4 presents the cost of RTEC$^{\rightarrow}$ in the worst-case, where we have an initiation/termination of an FVP at each time-point of the window. In practice, the number k of initiations and terminations of an FVP is much smaller than the window size ω, and thus the cost of RTEC$^{\rightarrow}$, i.e., $O(n_v klog(k))$, is significantly smaller. Note that the benefits of our approach, as compared to frameworks that lack the caching techniques of RTEC$^{\rightarrow}$, are considerable. The cost, e.g., of reasoning with rules (α)–(γ) is exponential in $\omega{-}R$. Below, we present an empirical analysis that illustrates the benefits of RTEC$^{\rightarrow}$.

6 Experimental Analysis

6.1 Experimental Setup

We evaluated RTEC$^{\rightarrow}$ in streaming applications that include events with delayed effects. From the field of multi-agent systems, we employed NetBill (Artikis and Sergot 2010) and a voting protocol (Pitt et al. 2006). For both NetBill and voting, we used synthetic event streams, simulating realistic multi-agent interactions. The task was to compute the normative positions of agents, such as permissions and obligations, as well as the sanctions imposed on agents over time. Furthermore, we evaluated RTEC$^{\rightarrow}$ on composite event recognition for maritime situational awareness. We used two real datasets containing position signals emitted from vessels sailing (a) in the area of Brest, France, over six months (5K vessels), and (b) in all European seas over one month (34K vessels). A description of both datasets may be found in (Pitsikalis et al. 2019). The task was to detect composite maritime activities, such as trawling and ship-to-ship transfer. We also evaluated RTEC$^{\rightarrow}$ on simulations of two biological 'feedback loops', i.e., control mechanisms expressing delayed changes in the values of biological variables, such as the concentration of an amino acid (Srinivasan et al. 2022). The task was to compute the values of the key variables as a simulation progressed. Our experiments are reproducible; the event descriptions of all applications and the corresponding datasets are available with the code of RTEC$^{\rightarrow}$[2]. The instructions for reproducing our experiments are in Appendix D.

We compared RTEC$^{\rightarrow}$ with the following computational frameworks that, although not optimised for handling events with delayed effects, are expressive enough to capture such events: RTEC$^{\rightarrow}$-naive, i.e., an implementation of RTEC$^{\rightarrow}$ that employs rules (α)–(γ) instead of the algorithms presented in Sections 4.2–4.3; s(CASP), i.e., a query-driven execution model for Answer Set Programming with constraints that has been extended with Event Calculus-based reasoning (Arias, Carro, Chen, et al. 2022; Arias, Carro, Salazar, et al. 2018); jREC$^{\text{fi}}$, i.e., an implementation of the 'Reactive Event Calculus' (Montali et al. 2013); and GKL-EC, i.e., a framework that integrates the Event Calculus with a sequential logic specifying asynchronous automata in order to process biological feedback loops (Srinivasan et al. 2022). Moreover, we employed the following frameworks that do not support events with delayed effects: Fusemate, i.e., a logic programming system that integrates the Event Calculus with description logics (Baumgartner 2021); jREC$^{\text{rbt}}$, i.e., a version of jREC that employs indexing for manipulating lists of events and FVP intervals efficiently (Falcionelli et al. 2019); Logica[3], i.e., a Datalog-like programming language that compiles into SQL and runs on BigQuery[4]; and Ticker, i.e., a stream reasoning system that supports a fragment of the logic-based language LARS (Beck, Dao-Tran, et al. 2018; Beck, Eiter, et al. 2017).

[2] https://github.com/aartikis/rtec
[3] https://github.com/EvgSkv/logica
[4] https://cloud.google.com/bigquery

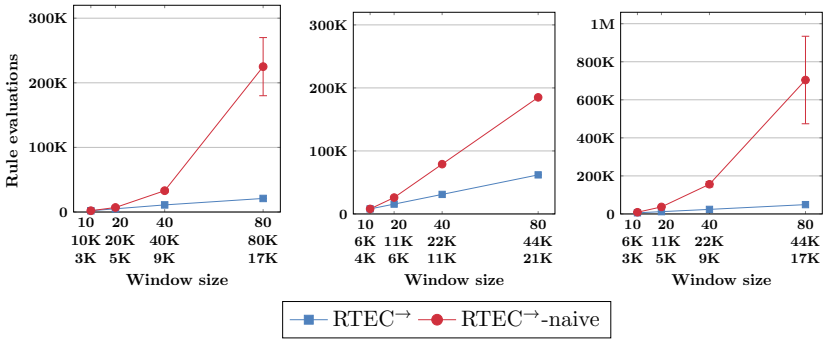

Fig. 4: Stream reasoning in voting with delayed effects that may not be postponed (left), and NetBill with delayed effects that may (middle) and may not be postponed (right). The horizontal axes display the window size in terms of time-points (top), avg. number of input events (middle) and avg. number of computed FVP intervals (bottom). The vertical axes show the avg. number of FVP rule evaluations.

Ticker is the only LARS-based framework in which we were able to express a meaningful subset of the event descriptions of RTEC$^{\rightarrow}$. A discussion of related research may be found in Section 7.

We ran each system using the programming language (version) recommended by its developers. Moreover, we made sure that all systems produced the same results. We used a single core of a desktop PC running Ubuntu 20.04, with Intel Core i7-4770 CPU @3.40GHz and 16GB RAM. For our evaluation on feedback loops, we performed an experiment for each possible configuration of initial variable values. For the remaining experiments, each result is the average of at least 30 queries. We report the standard deviations only when they are not negligible. Furthermore, in our comparisons of RTEC$^{\rightarrow}$ with related frameworks, we terminated the experiments that exceeded 15 minutes; we do not report the reasoning times of these experiments.

6.2 Experimental Results

In the first set of experiments, we compared RTEC$^{\rightarrow}$ against RTEC$^{\rightarrow}$-naive. Figure 4 shows the results of the comparison on voting and NetBill. Note that the event description of voting does not include future initiations that may be postponed. The window size ranged from 10 to 80 time-points, while the 'step', i.e., the distance between two consecutive query times, was set to 10 time-points. In NetBill, e.g., the windows contain, on average, 6K to 44K input events. We measured the number of rule evaluations of RTEC$^{\rightarrow}$ and RTEC$^{\rightarrow}$-naive per window, in order to assess the redundant computations performed by RTEC$^{\rightarrow}$-naive due to the use of rules (α)–(γ) instead of the optimised reasoning algorithms of RTEC$^{\rightarrow}$. The effects of such redundant rule evaluations on reasoning time increase with the number of conditions in the initiatedAt and the terminatedAt rules of the event description. Our results show that, as the window size increased, RTEC$^{\rightarrow}$-naive required a significantly larger number of rule computations to derive the same FVP intervals as RTEC$^{\rightarrow}$. The absence of a caching mechanism resulted in redundant rule computations, verifying our complexity analysis.

In the second set of experiments, we compared RTEC$^{\rightarrow}$ with other systems that support events with delayed effects. These systems, i.e., s(CASP), jREC$^{\text{fi}}$ and GKL-EC, do not support delayed effects that may postponed. First, we compared RTEC$^{\rightarrow}$ with s(CASP) and jREC$^{\text{fi}}$ on small subsets of voting and NetBill. The left diagram of Figure 5 shows the results of the comparison on voting; the results on NetBill are similar, and thus omitted. The window size and the step of RTEC$^{\rightarrow}$ were set to 10 time-points. s(CASP) and jREC$^{\text{fi}}$ do not use windows; they compute the values of fluents incrementally at each time-point. Figure 5 (left) presents the total reasoning times of RTEC$^{\rightarrow}$, s(CASP) and jREC$^{\text{fi}}$ for streams of increasing size. We observe that RTEC$^{\rightarrow}$ scales much better than

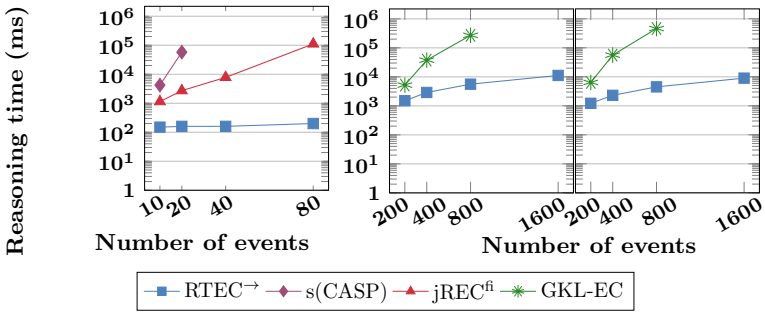

Fig. 5: Stream reasoning in a fragment of voting (left), and feedback loops for immune response (middle) and phage infection (right). The horizontal axes show the stream size; the vertical axes show the avg. of the total execution times.

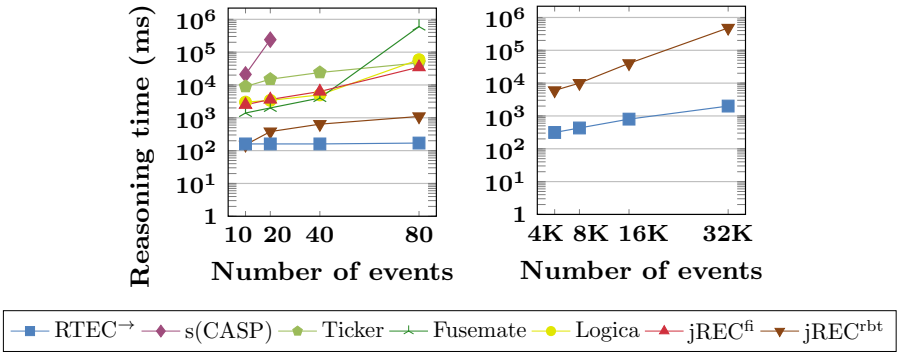

Fig. 6: Stream reasoning for computing *quote*.

s(CASP) and jREC^{fi}. s(CASP) performs point-based reasoning, as opposed to interval-based reasoning, which proved to be computationally expensive. jREC^{fi} and s(CASP) evaluated future initiations without caching previous derivations, which led to unnecessary re-computations. Moreover, jREC^{fi} and s(CASP) do not have a mechanism for 'forgetting' earlier events of the data stream, which makes them inappropriate for streaming applications. These performance issues did not allow us to perform larger-scale comparisons.

GKL-EC was developed specifically for processing biological feedback loops, and thus we assessed it only on this domain. The comparison concerns two feedback loops expressing immune response in vertebrates and the invasion of a phage in a bacteria cell (Srinivasan et al. 2022). In immune response, the presence of an antigen may lead to an increase in the production of antibodies after a certain delay. In phage invasion, the activation of a bacterial gene produces, after a delay, a protein that suppresses viral genes. We instructed RTEC$^{\rightarrow}$ and GKL-EC to compute the values of all variables in these loops as time progressed. The window size and the step of RTEC$^{\rightarrow}$ were set to 10 time-points. GKL-EC does not use windows. Figures 5 (middle and right) show the reasoning times of RTEC$^{\rightarrow}$ and GKL-EC for computing the values of loop variables over streams of increasing size. Our results demonstrate that RTEC$^{\rightarrow}$ scales significantly better than GKL-EC. GKL-EC performs point-based reasoning without 'forgetting', which hindered its performance.

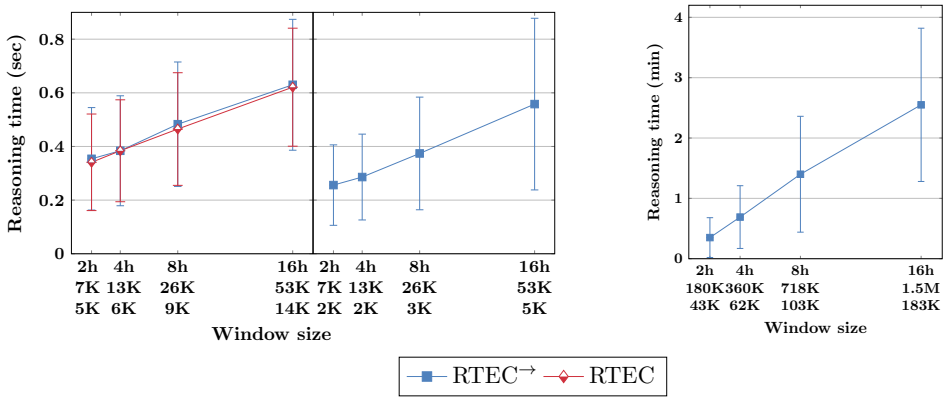

Fig. 7: Maritime situational awareness on the datasets of Brest for FVPs without future initiations (left) and FVPs with future initiations (middle), and all European seas for FVPs with future initiations (right). The horizontal axes display the window size in terms of temporal duration (top), avg. number of input events (middle) and avg. number of computed FVP intervals (bottom).

In the third set of experiments, we compared RTEC$^{\rightarrow}$ with related systems on a task that does not require future initiation computation. We instructed all systems to compute the maximal intervals of the *quote* fluent, as defined by rules (2) and (3), i.e., a specification of *quote* without future initiations. The comparison included s(CASP), jREC$^{\text{fi}}$, as well as systems without support for events with delayed effects, i.e., Ticker, Fusemate, Logica and jREC$^{\text{rbt}}$. Figure 6 (left) shows the total reasoning times over streams with size ranging from 10 to 80 events. The window size and the step of RTEC$^{\rightarrow}$ were set to 10 time-points, while the remaining systems do not employ windows. RTEC$^{\rightarrow}$ outperformed all frameworks, in most cases by several orders of magnitude. Ticker, Fusemate and Logica are not interval-based. jREC$^{\text{rbt}}$ scaled better than jREC$^{\text{fi}}$ thanks to FVP interval indexing. RTEC$^{\rightarrow}$ and jREC$^{\text{rbt}}$ were the most efficient systems, and thus we compared them further, using streams with thousands of events. Figure 6 (right) presents our results, demonstrating the benefits of RTEC$^{\rightarrow}$.

In the last set of experiments, we evaluated RTEC$^{\rightarrow}$ on maritime situational awareness. We used real streams containing millions of events that describe the activities of thousands of vessels. The volume of these streams did not allow for a comparison with the systems discussed earlier in this section. Figure 7 presents our results. The window size ranged from 2 to 16 hours and the step was set to 2 hours. Overlapping windows are necessary in the maritime domain due to the inherent lags in the arrival of vessel position signals, that may exceed 8 hours, especially concerning signals relayed through satellites (Pitsikalis et al. 2019). First, we compared RTEC$^{\rightarrow}$ and RTEC (Section 2) on the dataset concerning the area of Brest, France, on all FVPs except those subject to future initiations, since RTEC cannot handle events with delayed effects. This dataset includes positions signals emitted from approx. 5K vessels over a period of 6 months. The results of Figure 7 (left) indicate that our extension of RTEC does not compromise efficiency in temporal specifications without future initiations. Second, we evaluated RTEC$^{\rightarrow}$ on the detection of maritime activities specified by means of FVPs with future initiations, i.e., trawling and search and rescue operations. Figures 7 (middle and right) display the results on the datasets of Brest and all European seas, respectively. The latter dataset includes positions signals emitted from approx. 34K vessels over a period of one month. Our results show that RTEC$^{\rightarrow}$ is very efficient in challenging, real applications including events with delayed effects and complex temporal specifications. In the case of Brest, e.g., RTEC$^{\rightarrow}$ required, on average, less than 0.4 seconds to reason over the events of a 2-hour window, and approx. 0.6 seconds to process 16-hour windows, including 53K events, in order to compute 14K FVP intervals. In the case of all European seas,

RTEC$^{\rightarrow}$ reasoned over 2-hour windows in less than half a minute. For 16-hour windows, including approx. 1.5M events, RTEC$^{\rightarrow}$ required less than 3 minutes to compute 183K maximal intervals of FVPs.

7 Summary, Related and Further Work

We presented RTEC$^{\rightarrow}$, a framework for reasoning over streams of events with delayed effects. RTEC$^{\rightarrow}$ has a formal semantics, features an off-line reasoning process that enables the efficient execution of temporal specifications, avoids redundant run-time computations, and identifies the minimal information that should be transferred between windows to guarantee correctness. Our *reproducible* empirical evaluation on large synthetic and real data streams verified our complexity analysis, and demonstrated that RTEC$^{\rightarrow}$ outperforms the state-of-the-art by orders of magnitude.

RTEC$^{\rightarrow}$ is motivated by the work of Miller and Shanahan, where a first-order logic Event Calculus dialect was extended to support events with delayed effects (Miller and Shanahan 2002), and ECSTA, an Event Calculus State Tracking Architecture that uses 'timer fluents' to monitoring the delayed effects of events (Farrell et al. 2005). Several other formalisms support such events, including the action language presented in (Karlsson et al. 1998), Event Calculus-based formalisms (Marín and Sartor 1999), and multi-agent system specification languages expressing normative positions with deadlines (Chopra et al. 2020; Demolombe 2014; Hindriks and Riemsdijk 2013). There are several differences between RTEC$^{\rightarrow}$ and these approaches. Delayed effects that may be postponed, e.g., are not supported in (Chopra et al. 2020; Demolombe 2014; Karlsson et al. 1998; Marín and Sartor 1999; Miller and Shanahan 2002), while the formalisms presented in (Chopra et al. 2020; Demolombe 2014; Hindriks and Riemsdijk 2013; Marín and Sartor 1999; Miller and Shanahan 2002) do not support delayed effects that may lead to further delayed effects (see Example 3). Moreover, none the aforementioned approaches features reasoning techniques that are optimised for handling streaming applications.

Several systems for temporal pattern matching over event streams have been proposed in the literature (Bonte et al. 2024; Dell'Aglio et al. 2017; Fragkoulis et al. 2024). Implementations of LARS, i.e., the Laser reasoner (Bazoobandi et al. 2017), and the distributed reasoner presented in (Eiter et al. 2019), support negation only for globally stratified programs. StreamMill (Laptev et al. 2016) employs a minimal extension of SQL that supports stream reasoning. MeTeoR (Walega, Kaminski, Wang, et al. 2023) extends a fragment of DatalogMTL (Walega, Kaminski, and Grau 2019) with windowing. Ronca et al. studied the computational properties of streaming algorithms for a temporal extension of Datalog (Ronca, Kaminski, Grau, and Horrocks 2022; Ronca, Kaminski, Grau, Motik, et al. 2018). Similar to MeTeoR and StreamMill, this extension of Datalog does not support negation. Thus, these approaches cannot express the event descriptions of RTEC$^{\rightarrow}$. DDlog (Ryzhyk and Budiu 2019) is an incremental version of Datalog that handles continuously-arriving updates of input facts. DCDatalog (Wu et al. 2022) and BigDatalog (Shkapsky et al. 2016) are optimised with parallelisation techniques for recursive queries. DDLog, DCDatalog and BigDatalog reason over atemporal data, and thus cannot be employed directly for temporal reasoning tasks, such as evaluating the delayed effects of events.

RTEC$^{\rightarrow}$ may be used for composite event recognition, such as the detection of illegal maritime activities. There are several automata-based frameworks for composite event recognition (Demers et al. 2007; Poppe et al. 2019; Zhao et al. 2021). To facilitate the comparison of such frameworks, several works have proposed formal semantics for their temporal specification languages (Dindar et al. 2013; Grez and Riveros 2020; Grez, Riveros, and Ugarte 2019; Grez, Riveros, Ugarte, and Vansummeren 2021, 2020; Zielinski 2023). CORE, e.g., is an automata-based event recognition system with a formal semantics, that has proven highly efficient (Bucchi et al. 2022). However, CORE has limited expressivity, i.e., it supports only unary relations, applied only to the last event read. Increasing the expressivity of CORE by allowing higher arity relations undermines its worst-case time complexity. Wayeb, on the other hand, uses register automata to support n-ary relations (Alevizos et al. 2024). See (Cugola and Margara

2012; Giatrakos et al. 2020) for two surveys of event recognition systems. No such system is designed for handling events with delayed effects.

FVP hierarchies, i.e., FVPs appearing in the definitions of other FVPs, allow for structured, succinct representations, and thus code maintenance. A hierarchy of FVPs may exhibit a structure where FVPs participate in the definition of multiple other FVPs. RTEC$^{\rightarrow}$ can naturally handle hierarchies, paving the way for caching, and thus avoiding unnecessary re-computations. CORE and Wayeb, although exhibiting a compositional semantics, do not support hierarchies, in the sense that patterns are defined only over instantaneous properties. Point-based reasoning, as opposed to interval-based reasoning—see, e.g., ASTRO (Das et al. 2018) and the LARS reasoners Ticker and Laser (Beck, Dao-Tran, et al. 2018)—leads to unintended semantics when reasoning over hierarchies (Giatrakos et al. 2020; White et al. 2007).

D^{2}IA extends the streaming engine Flink with temporal operators, in order to reason over streams of durative events (Awad et al. 2022). Contrary to RTEC$^{\rightarrow}$, D^{2}IA does not support negation on durative phenomena. TPStream computes durative activities based on instantaneous events, and then matches temporal patterns over the derived activities (Körber et al. 2021). In contrast to RTEC$^{\rightarrow}$, TPStream cannot combine multiple event types in the definition of a durative activity. Moreover, D^{2}IA and TPStream do not allow background knowledge in event patterns. ISEQ performs temporal matching over streams of durative activities (Li et al. 2011); however, ISEQ cannot compute durative activities based on instantaneous events. ETALIS is a composite event recognition engine that supports patterns with durative activities and background knowledge (Anicic et al. 2012). ETALIS does not allow for the explicit representation of time, complicating the representation of the common-sense law of inertia for durative properties. Metric Compass Logic is a metric extension of Halpern-Shoham logic (Halpern and Shoham 1991) that has been used for inteval-based temporal event pattern matching (Ulus, Ferrère, et al. 2024; Ulus and Maler 2018). This logic, however, is proposotional, and thus does not capture the relational background knowledge and patterns that may be expressed in the language of RTEC$^{\rightarrow}$.

A key difference between RTEC$^{\rightarrow}$ and the aforementioned frameworks lies in the use of the Event Calculus (Kowalski and Sergot 1986), a logic programming formalism that is capable of expressing complex temporal patterns from a wide range of streaming applications, as it supports hierarchical and relational patterns, with spatio-temporal constraints and background knowledge. At the same time, the built-in representation of inertia allows us to develop succinct specifications, supporting code maintenance. The Event Calculus has been employed in numerous settings, including mobility assistance (Bromuri et al. 2010), reactive and proactive health monitoring (Chaudet 2006; Kafali et al. 2017), simulations with cognitive agents (Shahid et al. 2023, 2021) and computational ethics (Berreby et al. 2018). Contrary to the above approaches, RTEC$^{\rightarrow}$ is a formal framework that extends the Event Calculus with optimised streaming algorithms. RTEC$^{\rightarrow}$ is able to detect instances of complex temporal patterns over large streams from challenging, contemporary applications with minimal latency, outperforming the state-of-the-art.

The extensions of RTEC that have been proposed so far restrict attention to events with immediate effects. RTEC$_{\circ}$ supports cyclic dependencies among initiatedAt/terminatedAt rules (Mantenoglou, Pitsikalis, et al. 2022). RTEC$_{\mathrm{A}}$ supports Allen relations in FVP definitions (Mantenoglou, Kelesis, et al. 2023). RTEC$^{\rightarrow}$ combines and extends RTEC$_{\circ}$ and RTEC$_{\mathrm{A}}$ with support for events with delayed effects. RTEC$^{\rightarrow}$ benefits from the efficient cyclic dependency handling algorithms of RTEC$_{\circ}$, as they are used to support cyclic cd-components. RTEC$_{inc}$ performs rule re-writing to handle efficiently out-of-order streams (Tsilionis et al. 2022). RTEC$^{\rightarrow}$does not employ the rule re-writing of RTEC$_{inc}$. In the future, we aim to extend the techniques of RTEC$_{inc}$ for handling events with delayed effects. Moreover, we would like to extend RTEC$^{\rightarrow}$ in order to support fi relations between different fluents and conditional fi definitions.

S-Store is a framework that handles shared, mutable streams, ensuring data consistency in the presence of sequences of atomic transactions generated by distributed processes (Meehan et al. 2015; Tatbul et al. 2015). RIP employs a parallelisation technique in order to perform efficient temporal pattern matching over event

streams (Balkesen et al. 2013). RTEC$^{\rightarrow}$ assumes that the events in the input streams, that may stem from different processes, do not lead to inconsistencies, and does not employ parallelisation techniques; we leave the corresponding extensions of RTEC$^{\rightarrow}$ for future work.

Acknowledgments

We would like to thank Joaquín Arias, Peter Baumgartner, Stefano Bromuri, Thomas Eiter, Nicola Falcionelli, Gopal Gupta, Paul Ogris and Ashwin Srinivasan and for their help in the experimental comparisons. This work was supported by the EU-funded CREXDATA project (No 101092749).

References

E. Alevizos, A. Artikis, and G. Paliouras. 2024. "Complex Event Recognition with Symbolic Register Transducers." *Proc. VLDB Endow.*, 17, 11, 3165–3177.

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

J. Arias, M. Carro, Z. Chen, and G. Gupta. 2022. "Modeling and Reasoning in Event Calculus using Goal-Directed Constraint Answer Set Programming." *Theory Pract. Log. Program.*, 22, 1, 51–80.

J. Arias, M. Carro, E. Salazar, K. Marple, and G. Gupta. 2018. "Constraint Answer Set Programming without Grounding." *Theory Pract. Log. Program.*, 18, 3-4, 337–354.

M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia. 2018. "Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark." In: *SIGMOD*. ACM, 601–613.

A. Artikis and M. Sergot. 2010. "Executable Specification of Open Multi-Agent Systems." *Logic Journal of the IGPL*, 18, 1, 31–65.

A. Artikis, M. Sergot, and G. Paliouras. 2015. "An Event Calculus for Event Recognition." *IEEE Trans. Knowl. Data Eng.*, 27, 4, 895–908.

A. Artikis, M. Sergot, and J. Pitt. 2007. "An executable specification of a formal argumentation protocol." *Artif. Intell.*, 171, 10-15, 776–804. doi:10.1016/J.ARTINT.2007.04.008.

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

C. Balkesen, N. Dindar, M. Wetter, and N. Tatbul. 2013. "RIP: run-based intra-query parallelism for scalable complex event processing." In: *DEBS*. ACM, 3–14.

P. Baumgartner. 2021. "Combining Event Calculus and Description Logic Reasoning via Logic Programming." In: *FroCoS*. Vol. 12941, 98–117.

H. R. Bazoobandi, H. Beck, and J. Urbani. 2017. "Expressive Stream Reasoning with Laser." In: *ISWC*. Vol. 10587, 87–103.

H. Beck, M. Dao-Tran, and T. Eiter. 2018. "LARS: A Logic-based framework for Analytic Reasoning over Streams." *Artif. Intell.*, 261, 16–70.

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

F. Berreby, G. Bourgne, and J. Ganascia. 2018. "Event-Based and Scenario-Based Causality for Computational Ethics." In: *AAMAS*, 147–155.

P. Bonte et al.. 2024. "Grounding Stream Reasoning Research." *TGDK*, 2, 1, 2:1–2:47.

S. Bromuri, V. Urovi, and K. Stathis. 2010. "iCampus: A Connected Campus in the Ambient Event Calculus." *Int. J. Ambient Comput. Intell.*, 2, 1, 59–65.

M. Bucchi, A. Grez, A. Quintana, C. Riveros, and S. Vansummeren. 2022. "CORE: a COmplex event Recognition Engine." *Proc. VLDB Endow.*, 15, 9, 1951–1964.

H. Chaudet. 2006. "Extending the event calculus for tracking epidemic spread." *Artif. Intell. Medicine*, 38, 2, 137–156.

A. K. Chopra, S. H. Christie, and M. P. Singh. 2020. "An Evaluation of Communication Protocol Languages for Engineering Multiagent Systems." *J. Artif. Intell. Res.*, 69, 1351–1393.

K. Clark. 1978. "Negation as Failure." In: *Logic and Databases*. Plenum Press, 293–322.

G. Cugola and A. Margara. 2012. "Processing Flows of Information: From Data Stream to Complex Event Processing." *ACM Comput. Surv.*, 44, 3.

A. Das, S. M. Gandhi, and C. Zaniolo. 2018. "ASTRO: A Datalog System for Advanced Stream Reasoning." In: *CIKM*, 1863–1866.

D. Dell'Aglio, E. D. Valle, F. van Harmelen, and A. Bernstein. 2017. "Stream reasoning: A survey and outlook." *Data Sci.*, 1, 1-2, 59–83.

A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. M. White. 2007. "Cayuga: A General Purpose Event Monitoring System." In: *CIDR*, 412–422.

R. Demolombe. 2014. "Obligations with deadlines: a formalization in Dynamic Deontic Logic." *J. Log. Comput.*, 24, 1.

N. Dindar, N. Tatbul, R. J. Miller, L. M. Haas, and I. Botan. 2013. "Modeling the execution semantics of stream processing engines with SECRET." *VLDB J.*, 22, 4, 421–446.

T. Eiter, P. Ogris, and K. Schekotihin. 2019. "A Distributed Approach to LARS Stream Reasoning (System paper)." *Theory Pract. Log. Program.*, 19, 5-6, 974–989.
N. Falcionelli, P. Sernani, A. Brugués, D. N. Mekuria, D. Calvaresi, M. Schumacher, A. F. Dragoni, and S. Bromuri. 2019. "Indexing the Event Calculus: Towards practical human-readable Personal Health Systems." *Artif. Intell. Med.*, 96, 154–166.
A. D. H. Farrell, M. Sergot, M. Sallé, and C. Bartolini. 2005. "Using the event calculus for tracking the normative state of contracts." *Int. J. Cooperative Inf. Syst.*, 14, 2-3, 99–129. doi:10.1142/S0218843005001110.
M. Fragkoulis, P. Carbone, V. Kalavri, and A. Katsifodimos. 2024. "A survey on the evolution of stream processing systems." *VLDB J.*, 33, 2, 507–541.
N. Giatrakos, E. Alevizos, A. Artikis, A. Deligiannakis, and M. N. Garofalakis. 2020. "Complex event recognition in the Big Data era: a survey." *VLDB J.*, 29, 1, 313–352.
A. Grez and C. Riveros. 2020. "Towards Streaming Evaluation of Queries with Correlation in Complex Event Processing." In: *ICDT*. Vol. 155, 14:1–14:17.
A. Grez, C. Riveros, and M. Ugarte. 2019. "A Formal Framework for Complex Event Processing." In: *ICDT*. Vol. 127, 5:1–5:18.
A. Grez, C. Riveros, M. Ugarte, and S. Vansummeren. 2021. "A Formal Framework for Complex Event Recognition." *ACM Trans. Database Syst.*, 46, 4, 16:1–16:49.
A. Grez, C. Riveros, M. Ugarte, and S. Vansummeren. 2020. "On the Expressiveness of Languages for Complex Event Recognition." In: *ICDT*. Vol. 155, 15:1–15:17.
J. Y. Halpern and Y. Shoham. 1991. "A Propositional Modal Logic of Time Intervals." *J. ACM*, 38, 4, 935–962. doi:10.1145/115234.115351.
K. Hindriks and M. Riemsdijk. 2013. "A real-time semantics for norms with deadlines." In: *AAMAS*, 507–514.
Ö. Kafali, A. E. Romero, and K. Stathis. 2017. "Agent-oriented activity recognition in the event calculus: An application for diabetic patients." *Comput. Intell.*, 33, 4, 899–925.
L. Karlsson, J. Gustafsson, and P. Doherty. 1998. "Delayed Effects of Actions." In: *ECAI*, 542–546.
M. Körber, N. Glombiewski, A. Morgen, and B. Seeger. 2021. "TPStream: low-latency and high-throughput temporal pattern matching on event streams." *Distributed Parallel Databases*, 39, 2, 361–412.
R. Kowalski and M. Sergot. 1986. "A Logic-Based Calculus of Events." *New Gen. Computing*, 4, 1, 67–96.
N. Laptev, B. Mozafari, H. Mousavi, H. Thakkar, H. Wang, K. Zeng, and C. Zaniolo. 2016. "Extending Relational Query Languages for Data Streams." In: *Data Stream Management - Processing High-Speed Data Streams*. Data-Centric Systems and Applications. Springer, 361–386.
M. Li, M. Mani, E. A. Rundensteiner, and T. Lin. 2011. "Complex event pattern detection over streams with interval-based temporal semantics." In: *DEBS*. ACM, 291–302.
J. W. Lloyd. 1987. *Foundations of Logic Programming, 2nd Edition*. Springer.
P. Mantenoglou, D. Kelesis, and A. Artikis. 2023. "Complex Event Recognition with Allen Relations." In: *KR*.
P. Mantenoglou, M. Pitsikalis, and A. Artikis. 2022. "Stream Reasoning with Cycles." In: *KR*, 544–553.
R. H. Marín and G. Sartor. 1999. "Time and norms: a formalisation in the event-calculus." In: *ICAIL*. ACM, 90–99.
J. Meehan et al.. 2015. "S-Store: Streaming Meets Transaction Processing." *Proc. VLDB Endow.*, 8, 13, 2134–2145.
R. Miller and M. Shanahan. 2002. "Some Alternative Formulations of the Event Calculus." In: *Computational Logic: Logic Programming and Beyond*. LNAI 2408, 452–490.
M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. P. van der Aalst. 2013. "Monitoring business constraints with the event calculus." *ACM Trans. Intell. Syst. Technol.*, 5, 1, 17:1–17:30.
A. Paschke and M. Bichler. 2008. "Knowledge Representation Concepts for Automated SLA Management." *Decision Support Systems*, 46, 1, 187–205.
M. Pitsikalis, A. Artikis, R. Dreo, C. Ray, E. Camossi, and A. Jousselme. 2019. "Composite Event Recognition for Maritime Monitoring." In: *ACM DEBS*, 163–174.
J. Pitt, L. Kamara, M. Sergot, and A. Artikis. 2006. "Voting in Multi-Agent Systems." *Comput. J.*, 49, 2, 156–170.
O. Poppe, C. Lei, E. A. Rundensteiner, and D. Maier. 2019. "Event Trend Aggregation Under Rich Event Matching Semantics." In: *SIGMOD*. ACM, 555–572.
T. C. Przymusinski. 1988. "On the Declarative Semantics of Deductive Databases and Logic Programs." In: *Foundations of Deductive Databases and Logic Programming*. Ed. by J. Minker. Morgan Kaufmann, 193–216.
A. Ronca, M. Kaminski, B. C. Grau, and I. Horrocks. 2022. "The delay and window size problems in rule-based stream reasoning." *Artif. Intell.*, 306, 103668.
A. Ronca, M. Kaminski, B. C. Grau, B. Motik, and I. Horrocks. 2018. "Stream Reasoning in Temporal Datalog." In: *AAAI*, 1941–1948.
L. Ryzhyk and M. Budiu. 2019. "Differential Datalog." *Datalog*, 2, 4–5.
N. S. Shahid, D. O'Keeffe, and K. Stathis. 2023. "A Knowledge Representation Framework for Evolutionary Simulations with Cognitive Agents." In: *ICTAI*. IEEE, 361–368.
N. S. Shahid, D. O'Keeffe, and K. Stathis. 2021. "Game-theoretic Simulations with Cognitive Agents." In: *ICTAI*. IEEE, 1300–1305.

S. Sharifi, A. Parvizimosaed, D. Amyot, L. Logrippo, and J. Mylopoulos. 2020. "Symboleo: Towards a Specification Language for Legal Contracts." In: *28th IEEE International Requirements Engineering Conference, RE 2020, Zurich, Switzerland, August 31 - September 4, 2020*. Ed. by T. D. Breaux, A. Zisman, S. Fricker, and M. Glinz. IEEE, 364–369. doi:10.1109/RE48521.2020.00049.

A. Shkapsky, M. Yang, M. Interlandi, H. Chiu, T. Condie, and C. Zaniolo. 2016. "Big Data Analytics with Datalog Queries on Spark." In: *SIGMOD*, 1135–1149.

A. Srinivasan, M. Bain, and A. Baskar. 2022. "Learning explanations for biological feedback with delays using an event calculus." *Mach. Learn.*, 111, 7, 2435–2487.

N. Tatbul, S. Zdonik, J. Meehan, C. Aslantas, M. Stonebraker, K. Tufte, C. Giossi, and H. Quach. 2015. "Handling Shared, Mutable State in Stream Processing with Correctness Guarantees." *IEEE Data Eng. Bull.*, 38, 4, 94–104.

M. Tiger and F. Heintz. 2020. "Incremental reasoning in probabilistic Signal Temporal Logic." *Int. J. Approx. Reason.*, 119, 325–352.

E. Tsilionis, A. Artikis, and G. Paliouras. 2022. "Incremental Event Calculus for Run-Time Reasoning." *J. Artif. Intell. Res.*, 73, 967–1023.

D. Ulus, T. Ferrère, E. Asarin, D. Nickovic, and O. Maler. 2024. "Elements of Timed Pattern Matching." *ACM Trans. Embed. Comput. Syst.*, 23, 4, 59:1–59:45. doi:10.1145/3645114.

D. Ulus and O. Maler. 2018. "Specifying Timed Patterns using Temporal Logic." In: *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC 2018, Porto, Portugal, April 11-13, 2018*. Ed. by M. Prandini and J. V. Deshmukh. ACM, 167–176. doi:10.1145/3178126.3178129.

J. Verwiebe, P. M. Grulich, J. Traub, and V. Markl. 2023. "Survey of window types for aggregation in stream processing systems." *VLDB J.*, 32, 5, 985–1011.

P. A. Walega, M. Kaminski, and B. C. Grau. 2019. "Reasoning over Streaming Data in Metric Temporal Datalog." In: *AAAI*.

P. A. Walega, M. Kaminski, D. Wang, and B. C. Grau. 2023. "Stream reasoning with DatalogMTL." *J. Web Semant.*, 76.

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

J. Wu, J. Wang, and C. Zaniolo. 2022. "Optimizing Parallel Recursive Datalog Evaluation on Multicore Machines." In: *SIGMOD*.

B. Zhao, H. van der Aa, T. T. Nguyen, Q. V. H. Nguyen, and M. Weidlich. 2021. "EIRES: Efficient Integration of Remote Data in Event Stream Processing." In: *SIGMOD*. ACM, 2128–2141.

B. Zielinski. 2023. "Explanatory Denotational Semantics for Complex Event Patterns." *Formal Aspects Comput.*, 35, 4, 23:1–23:37.

Appendix Outline

Appendix A provides the reasoning algorithms of RTEC$^{\rightarrow}$ that were omitted from the paper. Appendix B contains complete proofs for the propositions concerning the correctness of RTEC$^{\rightarrow}$ and its complexity. Moreover, it outlines the cost of reasoning using rules (α)–(γ) of the paper, i.e., a naive implementation that is not used by RTEC$^{\rightarrow}$. Appendix C demonstrates that RTEC$^{\rightarrow}$ supports events that may lead to future terminations of domain properties. Appendix D provides instructions for reproducing the experiments of the empirical evaluation.

A Reasoning Algorithms

We provide the reasoning algorithms of RTEC$^{\rightarrow}$ that were omitted from the paper. First, we present the algorithm of RTEC$^{\rightarrow}$ for processing the attempt events that were cached at the previous query time (see line 2 of Algorithm 2 of the paper). Second, we present the algorithm that caches the attempt events that may lead to future initiations at the next query time (see line 8 of Algorithm 2). Third, we demonstrate how RTEC$^{\rightarrow}$ processes cd-components that contain cycles with f-edges (see line 13 of Algorithm 1 of the paper).

A.1 Processing Cached Attempt Events

Given $\mathsf{fi}(F=V, F=V', R)$, Algorithm 3 presents the steps for determining which one of the cached attempt events, that were generated by an initiation of $F=V$ and cached at the previous query time q_{i-1}, if any, may mark a future initiation of $F=V'$ at the current query time q_i. ω denotes the window size. $Attempts[F=V]$ is a sorted list containing the time-stamps of the attempt events that were generated by initiations of $F=V$ at q_{i-1}. $I_{q_{i-1}}[F=V]$ is a sorted list containing the intervals of $F=V$ that were computed by RTEC$^{\rightarrow}$ at q_{i-1}.

First, RTEC$^{\rightarrow}$ checks whether there is an interval in $I_{q_{i-1}}[F=V]$ that contains the beginning of the current window $q_i{-}\omega{+}1$ (see line 2 of Algorithm 3). If there is no such interval, then $F=V$ was 'broken' before $q_i{-}\omega{+}1$, and thus the cached attempts of initiating $F=V'$ have been cancelled. As a result, RTEC$^{\rightarrow}$ discards all cached

Algorithm 3 findAttempt

```
   Input: F=V, R
   Global: q_i, ω, Attempts[F=V], I_{q_{i-1}}[F=V]
   Output: T_att
1: T_att ← null
2: if q_i−ω+1 in I_{q_{i-1}}[F=V] then
3:     if p(F=V) then
4:         for each T in Attempts[F=V] do
5:             if T > q_i−ω and T−R ≤ q_i−ω then T_att ← T
6:             else if T−R > q_i−ω then return T_att
7:     else
8:         [T_s, T_f) ← getIntervalContainingTimepoint(I_{q_{i-1}}[F=V], q_i−ω+1)
9:         for each T_att in Attempts[F=V] do
10:            if T_att > q_i−ω and T_att−R == T_s then return T_att
11: return T_att
```

attempt events (see lines 1 and 11). If there is an interval in $I_{q_{i-1}}[F=V]$ that contains $q_i-\omega+1$, then $F=V$ holds up to $q_i-\omega+1$, and thus there may be a cached attempt of initiating $F=V'$ that is not cancelled. If the future initiations of $F=V'$ may be postponed, then RTEC$^{\rightarrow}$ returns the time-stamp T_{att} of the last attempt event *att* that takes place in the window, while the initiation of $F=V$ that generated *att* took place before the window (see lines 3–6). Earlier attempts of initiating $F=V'$ are postponed by the initiation of $F=V$ that generated attempt *att*, and are thus discarded. If the future initiations of $F=V'$ may not be postponed, then we keep the first attempt event *att* that was generated before the window and takes place in the window. *att* was produced by the initiation of $F=V$ that coincides with the starting point of the interval that contains $q_i-\omega+1$. Therefore, we retrieve that interval from $I_{q_{i-1}}[F=V]$ and keep the cached attempt event *att* that was generated by the initiation of $F=V$ that coincides with the starting point of the retrieved interval (see lines 7–10). Subsequent attempts of initiating $F=V'$ will be cancelled, at the latest, by T_{att}, i.e., the time-stamp of the derived attempt *att*, due to the potential future initiation of $F=V'$ at T_{att}.

A.2 Caching Attempt Events

Algorithm 4 shows the steps of RTEC$^{\rightarrow}$ for deriving, at q_i, the attempt events that may initiate $F=V'$, based on fi$(F=V, F=V', R)$, at the next query time q_{i+1}. The sorted lists $I[F=V]$ and $IP[F=V]$ contain, respectively, the intervals and the initiation points of $F=V$ that were derived at q_i.

We distinguish between two cases, depending on whether the future initiations of $F=V'$ may be postponed or not. If they may be postponed, then we have an attempt event R time-points after every initiation point of $F=V$ (see lines 2–4 of Algorithm 4). Otherwise, if the future initiations of $F=V'$ may not be postponed, then only the initiation points of $F=V$ that are starting points of maximal intervals of $F=V$ may induce a future initiation of $F=V'$. Any initiation point occurring within an interval of $F=V$ would not postpone the future initiation of $F=V'$ that was generated by the initiation of $F=V$ at the starting point of that interval, and thus does not need to be considered. Therefore, for each starting point T_s of an interval of $F=V$, we add time-point T_s+R in $Attempts[F=V]$ (see lines 5–7). RTEC$^{\rightarrow}$ caches the time-stamps in $Attempts[F=V]$, so that they will be available at the next query time (see line 8).

Algorithm 4 computeAttempts

Input: $F=V, R$
Global: $I[F=V], IP[F=V]$
Output: $Attempts[F=V]$
1: $Attempts[F=V] \leftarrow [\,]$
2: **if** p$(F=V)$ **then**
3: **for each** T **in** $IP[F=V]$ **do**
4: $Attempts[F=V]$.add$(T+R)$
5: **else**
6: **for each** $[T_s, T_f)$ **in** $I[F=V]$ **do**
7: $Attempts[F=V]$.add(T_s+R)
8: cache$(Attempts[F=V])$

A.3 Computing Future Initiations with Cycles

When processing a cd-component that contains a cycle with f-edges, RTEC$^{\rightarrow}$ employs an extended version of RTEC$_{\circ}$, featuring an algorithm for evaluating future initiations in the presence of cycles (see cyclicFl in line 13 of Algorithm 1 of the paper). For each fluent-value pair (FVP) $F=V$ in the cd-component, cyclicFl evaluates whether or not $F=V$ is initiated at each time-point T of the window using initiatedAtCyclic, i.e., an algorithm for evaluating an initiation of an FVP that is in a cd-component with cycles. When evaluating an initiatedAt rule defining $F=V$, initiatedAtCyclic does not assume that the maximal intervals of the FVPs appearing in body of the rule are certainly in the cache. Moreover, when evaluating a future initiation of $F=V$, based on, e.g., fi$(F=V^d, F=V, R)$, where $V^d \neq V$, initiatedAtCyclic does not assume that all the initiation points of $F=V^d$ that may lead to a future initiation of $F=V$ are in the cache. Additionally, initiatedAtCyclic does not assume that all the points that may cancel a future initiation of $F=V$ are cached.

Algorithm 5 presents initiatedAtCyclic. First, we make sure that time-point T falls within the current window (see line 1 of Algorithm 5). If so, RTEC$^{\rightarrow}$ checks whether the initiation of FVP $F=V$ at T has been evaluated at a previous step (see lines 2–3). If the cache of RTEC$^{\rightarrow}$ contains cachedInit$(F=V, T, +)$ (resp. cachedInit$(F=V, T, -)$), then it has been evaluated that $F=V$ is initiated (not initiated) at T. Otherwise, the initiation of $F=V$ at T has not been assessed at a previous step, and thus RTEC$^{\rightarrow}$ proceeds with its evaluation, starting with the initiatedAt rules of $F=V$ (see lines 4–7). The intervals of some FVPs appearing in the conditions of these rules may not be in the cache, due to cyclic dependencies in the definition of $F=V$. To address this issue, the truth values of holdsAt conditions are derived using RTEC$_{\circ}$. If an initiatedAt rule of $F=V$ is satisfied at T, then RTEC$^{\rightarrow}$ updates the cache with the derived initiation point (see line 6).

If an initiation of $F=V$ at T could not be computed based on the initiatedAt rules of $F=V$, then RTEC$^{\rightarrow}$ iterates over the fi facts that have $F=V$ as their second argument, in order to check whether one of them leads to a future initiation of $F=V$ at T (see line 8). Given the fact fi$(F=V^d, F=V, R)$, we have a future initiation of $F=V$ at T if (i) there is a cached attempt of initiating $F=V$, due to an earlier initiation of $F=V^d$, the time-stamp of the attempt, i.e., $AttemptOfFVP[F=V^d]$, coincides with T, and the future initiation of $F=V$ is not cancelled between the start of the window and T (see lines 9–12), or (ii) $F=V^d$ is initiated at $T-R$, and the future initiation of $F=V$ is not cancelled between $T-R$ and T (see lines 13–16). In both cases, RTEC$^{\rightarrow}$ consults the cache to avoid redundant evaluations of cancellation points (see cancelledCyclic in lines 10 and 14), and updates the cache with the derived initiation point, if any (see lines 11 and 15). If RTEC$^{\rightarrow}$ failed to compute an initiation point of $F=V$ at T, then we conclude that T is not an initiation point of $F=V$, and update the cache accordingly (see line 17).

Algorithm 5 initiatedAtCyclic($F=V$, T)

Input: q_i, ω, *AttemptOfFVP*.
Output: true/false.
1: **if** $T \leq q_i - \omega$ **or** $T > q_i$ **then return** false
2: **if** cachedInit($F=V$, T, +) **then return** true
3: **if** cachedInit($F=V$, T, −) **then return** false
4: **for each** *body* **in** bodiesOf(initiatedAt($F=V$, T)) **do**
5: **if** sat(*body*) **then**
6: updateCache(cachedInit($F=V$, T, +))
7: **return** true
8: **for each** fi($F=V^d$, $F=V$, R) **do**
9: **if** T==*AttemptOfFVP*[$F=V^d$] **then**
10: **if** not cancelledCyclic($F=V^d$, $q_i-\omega$, T) **then**
11: updateCache(cachedInit($F=V$, T, +))
12: **return** true
13: **if** $T > q_i-\omega+R$ **and** initiatedAtCyclic($F=V^d$, $T-R$) **then**
14: **if** not cancelledCyclic($F=V^d$, $T-R$, T) **then**
15: updateCache(cachedInit($F=V$, T, +))
16: **return** true
17: updateCache(cachedInit($F=V$, T, −))
18: **return** false

B Proofs of Correctness and Complexity

We provide complete proofs of Propositions 3 and 4 of the paper, which correspond to Propositions 5 and 6 of this document.

B.1 Correctness of RTEC$^{\rightarrow}$

Proposition 5 (Correctness of RTEC$^{\rightarrow}$): RTEC$^{\rightarrow}$ computes the maximal intervals of the FVPs of an event description, and no other interval. ▲

We show that Proposition 5 holds for an FVP $F=V$, such that fi($F=V$, $F=V'$, R), by proving the correctness of the novel operations of RTEC$^{\rightarrow}$. We assume that RTEC$^{\rightarrow}$ evaluates initiatedAt and terminatedAt rules (see lines 4–5 of Algorithm 1 of the paper) and constructs the maximal intervals of FVPs (see line 10 of Algorithm 1) correctly, because these operations are inherited from RTEC.

First, we prove that RTEC$^{\rightarrow}$ transfers the attempt events between windows that are required for correct FVP interval computation, and no other attempt event (see Lemma 1). Second, we show that RTEC$^{\rightarrow}$ computes all future initiations of $F=V'$, and no other future initiation (see Lemma 2). Third, at the time of constructing the maximal intervals of $F=V$, lists $IP[F=V]$ and $TP[F=V]$ contain the initiations and terminations of $F=V$, and no other point.

Lemma 1 (Correctness of transferring attempt events): RTEC$^{\rightarrow}$ transfers the attempt events between windows that are required for correct FVP interval computation, and no other attempt event. ▲

Proof. We prove that, at query time q_i, RTEC$^{\rightarrow}$ selects the attempt event that may mark a future initiation of $F=V'$, if any, as specified in the definition of fi (see Definition 5 of the paper). According to this definition, we have a future initiation of $F=V'$ at T_{att} iff there is an initiation of $F=V$ at $T_{att}-R$ and there is no cancellation

point of the future initiation of $F = V'$ between $T_{att}-R$ and T_{att}. Given window size ω, we will prove that RTEC$^{\rightarrow}$ derives an attempt event *att* iff its time-stamp T_{att} satisfies the following conditions:

(1) $T_{att} > q_i-\omega$.
(2) $T_{att}-R \leq q_i-\omega$.
(3) $F = V$ is initiated at $T_{att}-R$.
(4) The future initiation of $F = V'$ is not cancelled between $T_{att}-R$ and the start of the window $q_i-\omega+1$.
(5) The future initiation of $F = V'$ is not cancelled between $q_i-\omega+1$ and T_{att} by a future initiation of $F = V'$ that was generated before $q_i-\omega+1$.

Conditions 1 and 2 express that the attempt *att* falls inside the current window, while the initiation of $F = V$ that generated *att* is before the window. If this initiation of $F = V$ were also inside the window, then we would be able to perform the computation at q_i without transferring *att*. Conditions 3 and 4 express that the future initiation of $F = V'$ at T_{att} was staged and not cancelled up to $q_i-\omega+1$. The information before $q_i-\omega+1$ does not change at the q_i, and thus, if conditions 3 and 4 were violated at q_{i-1}, they are still violated at q_i. Condition 5 expresses that we should not select attempt event *att* if there is another attempt that satisfies conditions 1–4 and cancels the future initiation of $F = V'$ at T_{att}. This condition addresses the cases where we have several initiations of $F = V$ at $T_1, \ldots T_n$, where $T_n-T_1 < R$, and the future initiations of $F = V'$ may not be postponed. In such cases, only the attempt generated by the initiation of $F = V$ at T_1 is selected, because the remaining ones will be cancelled, at the latest, by the future initiation of $F = V'$ at T_1+R.

RTEC$^{\rightarrow}$ derives the attempt event that should be kept using Algorithm 3. Suppose that the output T_{att} of Algorithm 3 is not *null*, i.e., T_{att} is the time-stamp of an attempt event. If the future initiations of $F = V'$ may be postponed, then we have $T_{att} > q_i-\omega$ and $T_{att}-R \leq q_i-\omega$ (see line 4 of Algorithm 3). If the future initiations of $F = V'$ may not be postponed, then we have $T_{att} > q_i-\omega$, while $T_{att}-R$ coincides with the starting point of an interval that contains the start of the window (see lines 8 and 10), and thus $T_{att}-R \leq q_i-\omega$. In both cases, conditions 1 and 2 hold for T_{att}.

Next, we prove that $F = V$ is initiated at $T_{att}-R$. T_{att} is one of the time-stamps that were cached at previous query time by Algorithm 4 of RTEC$^{\rightarrow}$. If the future initiations of $F = V'$ may be postponed, then Algorithm 4 caches all time-points $T+R$, such that T is an initiation point of $F = V$. Otherwise, if the future initiations of $F = V'$ may not be postponed, then Algorithm 4 caches all time-points T_s+R, such that T_s is the starting point of an interval of $F = V$. Since all starting points of intervals of $F = V$ are initiation points of $F = V$, it holds that, if Algorithm 4 caches time-point T_s+R, then $F = V$ is initiated at T_s. As a result, all time-stamps in list $Attempts[F = V]$ of Algorithm 3 are R time-points after an initiation of $F = V$. Since the time-stamp T_{att} computed by Algorithm 3 is one of the items of list $Attempts[F = V]$, $T_{att}-R$ is an initiation point of $F = V$, and thus condition 3 holds.

T_{att} satisfies conditions 1–3. We prove that RTEC$^{\rightarrow}$ derives an attempt *att* with time-stamp T_{att} iff the future initiation of $F = V'$ is not cancelled between $T_{att}-R$ and the start of the window $q_i-\omega+1$ (condition 4), and it is not cancelled between $q_i-\omega+1$ and T_{att} by a future initiation of $F = V'$ that was generated before $q_i-\omega+1$ (condition 5). Assume that RTEC$^{\rightarrow}$ derives attempt *att*. Then, T_{att} is computed by Algorithm 3, meaning that there is an interval $[T_s, T_f)$ of $F = V$ that contains $q_i-\omega+1$ (see line 2 of Algorithm 3). The existence of interval $[T_s, T_f)$ implies that $F = V$ is not 'broken' between T_s and $q_i-\omega+1$, as, otherwise, $[T_s, T_f)$ would have been segmented. If the future initiations of $F = V'$ may not be postponed, then $T_{att}-R = T_s$ (see line 10). Therefore, the future initiation of $F = V'$ is not cancelled between $T_{att}-R$ and $q_i-\omega+1$, i.e., condition 4 holds, and it is not cancelled between $q_i-\omega+1$ and T_{att} by a future initiation of $F = V'$ that was induced earlier, because such a future initiation would have been cancelled before interval $[T_s, T_f)$, i.e., condition 5 holds. Otherwise, if the future initiations of $F = V'$ may be postponed, then $T_{att}-R$ coincides with the last initiation of $F = V$ in $[T_s, q_i-\omega+1)$ (see lines 5–6). Thus, $F = V$ is not 'broken' between $T_{att}-R$ and $q_i-\omega+1$, i.e., condition 4 holds, and earlier initiations of $F = V$ do not cancel the future initiation of $F = V'$ at T_{att}, as such earlier initiations would be postponed by the initiation of $F = V$

at $T_{att}-R$, i.e., condition 5 holds. Therefore, if RTEC$^{\rightarrow}$ computes an attempt event *att*, then its time-stampt T_{att} satisfies conditions 4 and 5.

Assume that T_{att} satisfies conditions 1–5. Based on condition 4, $F=V$ is not 'broken' between $T_{att}-R$ and $q_i-\omega+1$, and thus there is a maximal interval $[T_s, T_f)$ of $F=V$ that contains the start of the window $q_i-\omega+1$. If the future initiations of $F=V'$ may not be postponed, then RTEC$^{\rightarrow}$ computes T_s+R. Since $F=V$ holds continuously in $[T_s, T_f)$ and in $[T_{att}-R, T_f)$, and T_s is the starting point of a maximal interval, it holds that $T_{att}-R \geq T_s$. If $T_{att}-R > T_s$, then, since the future initiations of $F=V'$ may not be postponed, the future initiation of $F=V'$ that was induced at T_s will cancel the future initiation of $F=V'$ at T_{att}, meaning that T_{att} does not satisfy condition 5, which is a contraction. Therefore, we have $T_{att}-R=T_s$, and thus RTEC$^{\rightarrow}$ derives time-point T_{att}. If the future initiations of $F=V'$ may be postponed, then RTEC$^{\rightarrow}$ computes the time-stamp T of the last attempt of initiating $F=V'$ that was induced before $q_i-\omega+1$. Earlier attempts of initiating $F=V'$ are cancelled, because the corresponding future initiations of $F=V'$ are postponed by the initiation of $F=V$ that generated the attempt at T. Thus, these attempts do not satisfy condition 4, and the attempt event at T is only attempt that satisfies conditions 4 and 5. As a result, T coincides with T_{att}. Therefore, if T_{att} satisfies conditions 1–5, then RTEC$^{\rightarrow}$ derives the attempt event *att* that takes place at T_{att}.

We proved that RTEC$^{\rightarrow}$ derives an attempt event *att* iff its time-stamp T_{att} satisfies conditions 1–5. ◊

Lemma 2 (Correctness of future initiation computation): Given $\mathsf{fi}(F=V, F=V', R)$, RTEC$^{\rightarrow}$ computes all future initiations of $F=V'$, and no other future initiation. ▲

Proof. We start by presenting the proof for an acyclic cd-component containing the FVPs with fluent F; moreover, we assume that the vertex of FVP $F=V$ has no incoming f-edges, i.e., there are no future initiations of $F=V$. We will relax these assumptions in subsequent steps of the proof.

First, we prove that, at the time of evaluating the future initiations of $F=V'$ based on $\mathsf{fi}(F=V, F=V', R)$ (see line 7 of Algorithm 1 of the paper), the list $IP[F=V]$ of cached initiations of $F=V$ contains all initiations of $F=V$, and no other points, and the list $TP[F=V]$ of cached terminations of $F=V$ contains all terminations of $F=V$, except from those stemming from future initiations of $F=V'$, and no other points.

Correctness of $IP[F=V]$. $IP[F=V]$ contains the correct immediate initiations of $F=V$, because RTEC$^{\rightarrow}$ evaluated these initiations earlier, in line 4 of Algorithm 1, using an operation inherited from RTEC, which we have assumed to be correct. Since there are no future initiations of $F=V$, we conclude that $IP[F=V]$ contains all initiations of $F=V$, and no other points.

Correctness of $TP[F=V]$. $TP[F=V]$ contains the correct immediate terminations of $F=V$, because these terminations were computed at an earlier step by RTEC (see line 5 of Algorithm 1). $TP[F=V]$ does not contain future terminations of $F=V$ that stem from future initiations of an FVP $F=V_j$, where $V_j \neq V'$, as such future initiations do not terminate $F=V$. Suppose that we have $\mathsf{fi}(F=V_i, F=V_j, R')$, where $V_i \neq V_j \neq V$ and $V_j \neq V'$. (We assume that $V_i \neq V$ without loss of generality, because there may be, at most, one fi with $F=V$ as its first argument. If there were a fact $\mathsf{fi}(F=V, F=V'', R'')$, where $R'' > R$, then such a fact would never result in a future initiation of $F=V''$.) In order for a future initiation of $F=V_j$ to not get cancelled, $F=V_i$ must hold up to the time-point of the future initiation. Since fluents have at most one value at a time, it is not possible for $F=V$ to hold at the time of the future initiation of $F=V_j$. Therefore, a future initiation of $F=V_j$ cannot terminate $F=V$. As a result, the future initiations of $F=V'$ are the only future initiations of F that may terminate $F=V$. Therefore, the only terminations of $F=V$ that are missing from $TP[F=V]$, if any, are the ones stemming from future initiations of $F=V'$.

Second, we prove that, given lists $IP[F=V]$ and $TP[F=V]$, RTEC$^{\rightarrow}$ derives all the future initiations of $F=V'$, and no other time-points. According to Definition 5, there is a future initiation of $F=V'$ based on $\mathsf{fi}(F=V, F=V', R)$ at $T+R$ iff $F=V$ is initiated at T and there is no cancellation point of the future initiation of $F=V'$ between T and $T+R$.

Correctness of future initiation evaluation at T_{att}. Suppose that there is a cached attempt event *att* with timestamp T_{att}. RTEC$^{\rightarrow}$ computes a future initiation of $F = V'$ at T_{att} iff there is no cached cancellation point of the future initiation of $F = V'$ between the start of the window $q_i - \omega + 1$ and T_{att} (see lines 3–4 of Algorithm 2 of the paper). Since this is the earliest potential future initiation of $F = V'$ at q_i, $TP[F = V]$ contains all the termination points of $F = V$ that are between $q_i - \omega + 1$ and T_{att}, and no other points. Therefore, the cache contains all cancellation points of future initiations of $F = V'$ between $q_i - \omega + 1$ and T_{att}, and no other points. Moreover, there is no cancellation point between $T_{att} - R$ and T_{att} that was generated before $q_i - \omega + 1$ (see Lemma 1). Therefore, RTEC$^{\rightarrow}$ computes a future initiation at T_{att} iff there is no cancellation point between $T_{att} - R$ and T_{att}.

Correctness of future initiation evaluation after T_{att}. Suppose that $T_1, \ldots, T_k$ are the temporally sorted initiations of $F = V$. We show that RTEC$^{\rightarrow}$ computes a future initiation of $F = V'$ at $T_i + R$, where $1 \leq i \leq k$, iff the future initiation of $F = V'$ is not cancelled between T_i and $T_i + R$. For the base case, according to lines 5–7 of Algorithm 2 of the paper, RTEC$^{\rightarrow}$ computes a future initiation of $F = V'$ at $T_1 + R$ iff there is no cancellation point between T_1 and $T_1 + R$. Note that there are no initiations of $F = V$ before T_1, and thus no future initiations of $F = V'$ before $T_1 + R$. In other words, the cache has all cancellation points of the future initiation of $F = V'$ between T_1 and $T_1 + R$. Next, assume that RTEC$^{\rightarrow}$ has evaluated correctly the future initiations of $F = V'$ up to $T_{n-1} + R$, where $1 < n \leq k$. Since RTEC$^{\rightarrow}$ caches the future initiations that it derives (line 7 of Algorithm 2), the cache contains all cancellation points of the future initiations of $F = V'$ up to $T_{n-1} + R$. As a result, RTEC$^{\rightarrow}$ computes a future initiation of $F = V'$ at $T_n + R$ iff there is no cancellation point between T_n and $T_n + R$.

Third, we generalise the proof for the case where the vertex of $F = V$ has incoming f-edges, i.e., we may have future initiations of $F = V$. In this case, we may have additional initiations of $F = V$, and thus we have to re-establish the correctness of $IP[F = V]$.

Correctness of $IP[F = V]$ with future initiations of $F = V$. We consider the case where the vertex of $F = V$ has incoming f-edges, i.e., we may have future initiations of $F = V$, and prove that $IP[F = V]$ contains all initiations of $F = V$, and no other points. Suppose that we have n facts $\mathsf{fi}(F = V_i, F = V, R_i)$, where $1 \leq i \leq n$, and all values V_i are pairwise different. Since there are no cycles in the dependency graph, RTEC$^{\rightarrow}$ processes FVPs $F = V_1, \ldots, F = V_n$ before $F = V$ (see Definition 12 of the paper). For each FVP $F = V_i$, if we may not have future initiations of $F = V_i$, then RTEC$^{\rightarrow}$ evaluates the future initiations of $F = V$ based on $\mathsf{fi}(F = V_i, F = V, R_i)$ correctly, as proven above, and stores them in list $IP[F = V]$ (see lines 7 and 9 of Algorithm 1). Otherwise, if there are some fi facts defining $F = V_i$, then, based on the acyclicity of the cd-component, we may use an inductive proof on the vertices of the cd-component, in order to show that the future initiations of $F = V$ based on $\mathsf{fi}(F = V_i, F = V, R_i)$ are derived correctly. As a result, when processing FVP $F = V$, $IP[F = V]$ contains all the future initiations of $F = V$.

Fourth, we generalise the proof for the case where we have a cd-component with cycles. In this case, we show that future initiation evaluation is reduced to initiatedAt/terminatedAt rule evaluations, which are inherited from RTEC, in conjunction with caching intermediate derivations.

Correctness of future initiation evaluation with cycles. Suppose that there is a cycle of f-edges that includes f-edge $(v_{F=V}, v_{F=V'})$. In this case, RTEC$^{\rightarrow}$ does not employ Algorithm 2 of the paper; instead, it uses Algorithm 5 to evaluate the initiations of all FVPs in the cycle. When evaluating the future initiations of $F = V'$, Algorithm 5 does not assume that all initiations of $F = V$ and all cancellation points of future initiations of $F = V'$ have been evaluated and cached. To address this issue, Algorithm 5 follows a direct implemenation of the definition of fi (see Definition 5 of the paper), according to which any initiation/termination point that is required to evaluate a future initiation of $F = V'$ and has not been evaluated in a previous step, based on the cache, is determined using initiatedAt/terminatedAt rules. As a result, the correctness of Algorithm 5 follows from the correctness of rule evaluation, which we have assumed to be correct. Therefore, RTEC$^{\rightarrow}$ evaluates correctly future initiations with cycles. ◊

In the proof of Lemma 2, we demonstrated that, at the time of evaluating the future initiations of $F=V'$, the cache contains all initations of $F=V$, and no other initiation points, and all terminations of $F=V$, except from those stemming from future initiations of $F=V'$, and no other termination points. Moreover, we proved that RTEC$^{\rightarrow}$ computes the future initiations of $F=V'$, and no other points. According to line 8 of Algorithm 1 of the paper, the derived future initiations of $F=V'$ are added in the list of cached terminations of $F=V$. As a result, at the time of constructing the maximal intervals of $F=V$, the cache contains the initiations and terminations of $F=V$, and no other points, leading to correct maximal interval construction. Therefore, RTEC$^{\rightarrow}$ computes the maximal intervals of all FVPs of an event description, and no other interval.

B.2 Complexity of RTEC$^{\rightarrow}$

Proposition 6 (Complexity of RTEC$^{\rightarrow}$): The cost of evaluating the future initiations of all FVPs in a cd-component is $O(n_v(\omega-R)log(\omega))$, where n_v is the number of possible values of an FVP, ω is the window size and R is the delay of a future initiation. ▲

First, we demonstrate the complexity of transferring attempt events between windows (see Lemma 3). Second, we outline the cost of evaluating the future initiations of all FVPs in an acyclic cd-component (see Lemma 4). Third, we discuss the case of a cd-component with cycles (see Lemma 5).

Lemma 3 (Complexity of transferring attempt events): The cost of transferring attempt events between windows is $O(\omega)$. ▲

Proof. Consider fi$(F=V, F=V', R)$. RTEC$^{\rightarrow}$ employs Algorithm 4 to determine which attempt events of initiating $F=V'$ will be cached. In the worst case, Algorithm 4 iterates over the list of cached initiation points of $F=V$ once. The size of this list is bounded by the window size ω, and thus the worst case complexity of Algorithm 4 is $O(\omega)$. RTEC$^{\rightarrow}$ employs Algorithm 3 to determine which one the cached attempt events, if any, may mark a future initiation of $F=V'$. In the worst case, Algorithm 3 iterates over the list of cached attempt events once, and the number of these events is bounded by the window size. As a result, the cost of Algorithm 3 is $O(\omega)$. Therefore, the cost transferring attempt events between windows is $O(\omega)$. ◊

Lemma 4 (Complexity of processing an acyclic cd-component): The cost of evaluating the future initiations of all FVPs in an acyclic cd-component is $O(n_v(\omega-R)log(\omega))$, where n_v is the number of possible values of an FVP, ω is the window size and R is the delay of a future initiation. ▲

Proof. Consider fi$(F=V, F=V', R)$ and an an acyclic cd-component that contains the vertex of FVP $F=V$. We compute the cost of evaluating the future initiations of $F=V'$. For each initiation point T of $F=V$, where $T \leq q_i-R$, we may have a future initiation of $F=V'$ that falls inside the window. The number of these initiations is bounded by $\omega-R$. Moreover, there may be a future initiation of $F=V'$ at the time of the cached attempt event, if any. Therefore, RTEC$^{\rightarrow}$ may evaluate a future initiation of $F=V'$ at most $\omega-R+1$ times. To evaluate each future initiation of $F=V'$, RTEC$^{\rightarrow}$ performs a cache retrieval operation from the sorted lists of initiation and termination points of $F=V$, in order to check whether one of them cancels the future initiation of $F=V'$. The size of each list is bounded by ω, and thus the cost of determining whether a future initiation is cancelled is $O(log(\omega))$. Therefore, after simplifications, the cost of evaluating the future initiations of $F=V'$ is $O((\omega-R)\log(\omega))$. In an acyclic cd-component, all vertices are connected with f-edges, and thus correspond to FVPs with the same fluent. As a result, an acyclic cd-component may have at most n_v FVPs. Thus, the cost of evaluating the future initiations of all FVPs in a cd-component is $O(n_v(\omega-R)log(\omega))$. ◊

Lemma 5 (Complexity of processing a cd-component with cycles): The cost of evaluating the future initiations of all FVPs in an cd-component with cycles is $O(n_v(\omega-R)log(\omega))$, where n_v is the number of possible values of an FVP, ω is the window size and R is the delay of a future initiation. ▲

Proof. Consider $\mathsf{fi}(F=V, F=V', R)$ and that the cd-component that includes the vertices of $F=V$ and $F=V'$ contains cycles with f-edges. In this case, RTEC$^{\rightarrow}$ does not assume that the cancellation points of future initiations of $F=V'$ are available in the cache. To address this issue, RTEC$^{\rightarrow}$ employs Algorithm 5, which may, at most, evaluate the future initiation of each FVP in the cd-component once, at each of the $\omega{-}R{+}1$ time-points of the window where we may have such an initiation. This is because Algorithm 5 uses the cache of RTEC$^{\rightarrow}$ in order to store previous evaluation of FVP initiations, and thus avoids re-computations. As a result, the cost of evaluating the future initiations of all FVPs in a cd-component including cycles with f-edges remains $O((\omega{-}R)n_v log(\omega))$. ♢

Based on Lemmata 3, 4 and 5, the worst-case cost of evaluating the future initiations of all FVPs in a cd-component, including windowing, is $O(n_v(\omega{-}R)log(\omega))$.

Proposition 6 presents the cost of RTEC$^{\rightarrow}$ in the worst-case, where we have an initiation/termination of an FVP at each time-point of the window. In practice, the number k of initiations and terminations of an FVP is much smaller than the window size ω, resulting in fewer computations than in the worst-case. Given an acyclic cd-component with $\mathsf{fi}(F=V, F=V', R)$, for each initiation T of $F=V$, RTEC$^{\rightarrow}$ evaluates a future initiation of $F=V'$ at $T{+}R$, resulting in k evaluations. In order to decide whether a future initiation of $F=V'$ is cancelled, RTEC$^{\rightarrow}$ examines a cache of at most k points, leading to a cost of $O(log(k))$. Moreover, the number of cached attempt events is bounded by k, and thus the cost of windowing becomes $O(k)$. As a result, when $k \ll \omega$, the cost of evaluating the future initiations of all FVPs in a cd-component becomes $O(n_v k log(k))$, which is signicantly smaller compared to the worst-case. If the cd-component has cycles, then we have to evaluate future initiations of $F=V'$ at $\omega{-}R$ time-points, because some initiations of $F=V$ may not be in the cache. Moreover, the cache contains the results of earlier evaluations of future initiations of $F=V'$ at each time-point of the window, and thus the cost of retrieving a point from the cache is $O(log(\omega))$. Therefore, in the case of cycles with f-edges, the cost of future initiation evaluation remains $O((\omega{-}R)n_v log(\omega))$, which shows that, in practice, RTEC$^{\rightarrow}$ processes cd-components without cycles more efficiently than cd-components with cycles, thanks to its processing order of FVPs.

B.3 Complexity of Rules (α)–(γ)

Rules (α)–(γ) of the paper can be used to evaluate future initations of FVPs. However, these rules are not optimised for reasoning over data streams, and thus they are not part of RTEC$^{\rightarrow}$. We compare RTEC$^{\rightarrow}$ with rules (α)–(γ), in order to highlight the benefits of the processing order and the caching mechanism of RTEC$^{\rightarrow}$. Below, we outline the worst-case complexity of rules (α)–(γ).

Proposition 7: The cost of evaluating future initiations using rules (α)–(γ) of the paper is $O((n_v R)^{\omega-R})$, where n_v is the number of possible values of an FVP, ω is the window size and R is the delay of a future initiation. ▲

Proof. Suppose that we have $\mathsf{fi}(F=V, F=V', R)$, and the time-points of the window are $T_1, T_2, \ldots, T_\omega$. $\mathsf{fiCost}(T_n)$ denotes the cost of evaluating a future initiation of $F=V'$ at T_n and m denotes the cost of computing whether an FVP is initiated or terminated at some time-point based on initiatedAt/terminatedAt rules. Based on rule (α), we have:

$$\mathsf{fiCost}(T_n) = \begin{cases} m{+}\mathsf{fiCost}(T_{n-R}){+}cc(T_{n-R}, T_n) & n > R \\ cc(T_1, T_n) & 1 \leq n \leq R, \mathsf{holdsAt}(F=V, T_1), T_n = T_{att} \\ O(1) & otherwise \end{cases}$$

If T_n is after T_R, then there may be a future initiation of $F=V'$ at T_n which is induced by an initiation of $F=V$ at the time-point T_{n-R} of the current window. The cost of evaluating a future initiation of $F=V'$ at T_n is equal to the cost of evaluating whether $F=V$ is initiated at T_{n-R}, i.e., the cost m of evaluating its initiatedAt rules plus the cost $\mathsf{fiCost}(T_{n-R})$ of evaluating whether there is a future initiation of $F=V$ at T_{n-R}, plus the cost $cc(T_{n-R}, T_n)$ of checking whether the future initiations of $F=V'$ is cancelled between T_{n-R} and T_n. If T_n is not after T_R, then there

may be a future initiation of $F = V'$ at T_n only if T_n coincides with the time-stamp T_{att} of the derived attempt event *att*. In this case, the cost of $\mathsf{fiCost}(T_n)$ is equal to the cost of $cc(T_1, T_n)$. Otherwise, $\mathsf{fiCost}(T_n)$ simply returns false in $O(1)$.

The cost of evaluating $cc(T_{n-R}, T_n)$ is:

$$cc(T_{n-R}, T_n) = \sum_{i=n-R+1}^{n-1} (m + \sum_{V^d \neq V} (m + \mathsf{fiCost}(T_i))) \tag{4}$$

For each time-point T_i between T_{n-R} and T_n, we compute whether $F = V$ is terminated at T_i with cost m and, then, for each value V^d of F other than V, we compute whether $F = V^d$ is initiated at T_i based on initiatedAt rules and whether there is a future initiation of $F = V^d$ at T_i.

Because of the caching and indexing techniques of RTEC$^{\rightarrow}$, the cost m remains close to constant. Thus, we omit it from the formulas below. After substituting the cost of $cc(T_{n-R}, T_n)$, according to equation (4), in the definition of $\mathsf{fiCost}(T_n)$, and disregarding the cost m, the cost $\mathsf{fiCost}(T_n)$, where $n > R$, is:

$$\mathsf{fiCost}(T_n) = \mathsf{fiCost}(T_{n-R}) + \sum_{i=n-R+1}^{n-1} \sum_{V^d \neq V} \mathsf{fiCost}(T_i)$$

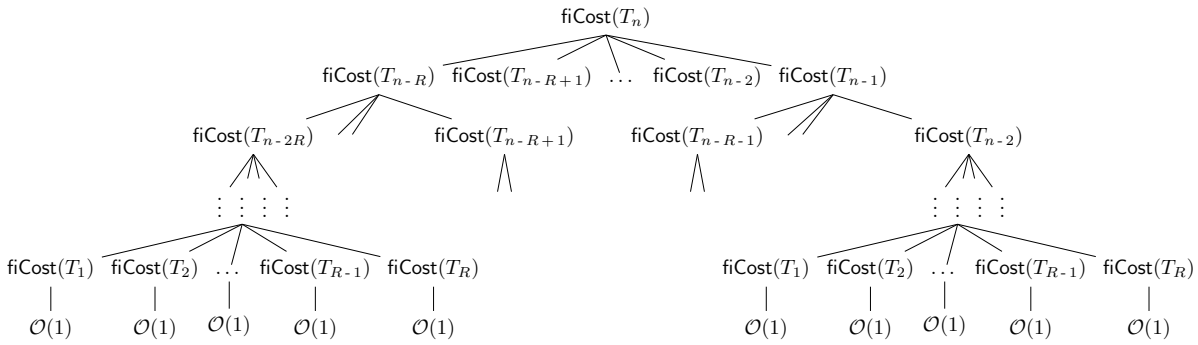

Fig. 8: Recursion tree for evaluating a future initiation at time-point T_n, where $n > R$.

Figure 8 shows the recursion tree for $\mathsf{fiCost}(T_n)$, in the simple case where F has two possible values. Suppose that F has n_v possible values. Then, each node $\mathsf{fiCost}(T_n)$, where $n > R$, has $1 + n_v(R-1)$ children nodes. Moreover, the height of the tree is $\omega - R$. This can be verified in the case of $n = \omega$ by following the rightmost child of each node of the recursion tree, until reaching $\mathsf{fiCost}(T_R)$, which does not have any children. Therefore, after simplifications, the cost of evaluating $\mathsf{fiCost}(T_n)$ with rules (α)–(γ) is $O((n_v R)^{\omega - R})$. ◊

C Future Terminations

RTEC$^{\rightarrow}$ also supports events that may terminate an FVP in the future. We demonstrate that minor extensions to the syntax and the reasoning algorithms of the paper are sufficient to express and reason over such events. The semantics of RTEC$^{\rightarrow}$ are not affected by these extensions; future terminations do not introduce additional dependencies among FVPs, and thus the semantics of RTEC$^{\rightarrow}$ remains locally stratified logic programs.

C.1 Syntax

An event description in RTEC$^{\rightarrow}$ may contain facts expressing future terminations of FVPs.

Definition 18 (Event Description in RTEC$^{\rightarrow}$): We extend Definition 5 of the paper with the following type of facts:

- ft$(F=V,R)$, where R is a positive integer, expressing that the initiation of $F=V$ at a time-point T leads to the ***f**uture **t**ermination* (ft) of $F=V$ at time-point T+R, provided that $F=V$ is not 'broken' between T and T+R. ■

ft facts express a different type of delayed effects compared to fi facts. Suppose that $F=V$ is initiated at time-point T, and that the delayed effects of events are not cancelled. According to ft$(F=V,R)$, there is a future termination of $F=V$ at time-point T+R, which implies that we become agnostic about the value of F after T+R. Contrast this with the effects of an fi fact, e.g., fi$(F=V,F=V',R)$, where $V\neq V'$, according to which there is a future initiation of $F=V'$ at time-point T+R, meaning that the value of F is V' after T+R.

C.2 Reasoning

Since ft facts do not affect the dependency graph of RTEC$^{\rightarrow}$, we do not need to modify the processing order relation presented in the paper. Below, we show the reasoning algorithms of RTEC$^{\rightarrow}$ that incorporate future terminations. Algorithm 6 is a modified version of Algorithms 1 of the paper that processes ft facts. Algorithm 7 shows the steps of RTEC$^{\rightarrow}$ for computing future terminations.

Algorithm 6 processCDComponent

Input: cd-component G_{S_i}, cached intervals I.
Output: I, including the intervals of the FVPs in G_{S_i}.

```
1:  if G_Si does not contain a cycle then
2:      for each v_F=V in G_Si do IP[F=V] ← [ ]
3:      for each v_F=V in processingOrder(G_Si) do
4:          IP[F=V].add(RTEC.evalIP(F=V, I))
5:          TP[F=V]←RTEC.evalTP(F=V, I)
6:          if fi(F=V, F=V', R) then
7:              FtIP[F=V']←evalFI(IP[F=V], TP[F=V], V', R)
8:              TP[F=V].add(FtIP[F=V'])
9:              IP[F=V'].add(FtIP[F=V'])
10:         else if ft(F=V, R) then
11:             TP[F=V].add(evalFT(IP[F=V], TP[F=V], R))
12:         I[F=V]←RTEC.mi(IP[F=V], TP[F=V])
13: else if G_Si does not contain an f-edge then
14:     I.updateIntervals(RTEC_∘(G_Si, I))
15: else I.updateIntervals(cyclicFI(G_Si, I))
16: return I
```

If we have an fi fact and an ft fact with the same FVP $F=V$ as their first argument, then only the one with the shorter delay will be meaningful, while the other fact will not generate any results, regardless of the input stream. Therefore, RTEC$^{\rightarrow}$ considers that $F=V$ may be the first argument of either an fi fact or an ft fact (see lines 6 and 10 of Algorithm 6). When we have ft$(F=V,R)$, RTEC$^{\rightarrow}$ computes the future terminations of $F=V$ and adds them in list $TP[F=V]$, which is maintained temporally sorted (see lines 10–11). The reasoning steps of

RTEC$^{\rightarrow}$ for computing the future terminations of $F=V$ (see Algorithm 7) are the same as the ones presented in Algorithm 2 of the paper for computing future initiations.

Algorithm 7 evalFT

Input: $IP[F=V]$, $TP[F=V]$, R.
Global: q_i, ω.
Output: The future terminations of $F=V$.

$FtTP[F=V] \leftarrow [\,]$
$T_{att} \leftarrow$ findAttempt$(F=V, R)$
if $T_{att} \neq null$ **and** not cancelled$(q_i-\omega, T_{att}, IP[F=V], TP[F=V])$ **then**
  $FtTP[F=V]$.add(T_{att})
for each T **in** $IP[F=V]$ **do**
  if $T+R \leq q_i$ **and** not cancelled$(T, T+R, IP[F=V], TP[F=V] \cup FtTP[F=V])$ **then**
    $FtTP[F=V]$.add$(T+R)$
computeAttempts$(F=V, R)$
return $FtTP[F=V]$

As demonstrated above, RTEC$^{\rightarrow}$ supports future terminations of FVPs with only minimal changes in its reasoning algorithms. The correctness proof of RTEC$^{\rightarrow}$ can be modified to incorporate future terminations with only minor extensions. Moreover, the cost of RTEC$^{\rightarrow}$ does not increase with the introduction of ft facts; the tasks of evaluating fi and ft facts have the same worst-case complexity.

D Reproducing Our Experiments

The code of RTEC$^{\rightarrow}$ is publicly available[2]. We provide the code for reproducing our experiments in the form of a Docker image, available in a github repository[5]. Below, we describe the steps for setting up the image on Ubuntu Linux. See the online documentation of Docker[6] for the syntax of the required Docker commands in other operating systems.

Download and save our zipped code in the directory of your choice. Unzip the archive as follows:

```
cd directory/of/zip/file
unzip rtec_experiments.zip -d rtec_experiments
cd rtec_experiments
```

Install Docker[7], and then follow these steps:

```
sudo docker build -t experiments . # Build a Docker image based on the Dockerfile.
sudo docker run -it experiments # Run the Docker image in interactive mode.
./run_all_experiments.sh # Execute the script that runs all experiments.
```

To reproduce a specific set of experiments, run the corresponding execution script as follows:

```
cd scripts
ls # See the available scripts.
./name_of_chosen_script.sh
```

You may remove the Docker image from your system as follows:

[5] https://github.com/Periklismant/rtec_experiments
[6] https://docs.docker.com/
[7] https://docs.docker.com/engine/install/ubuntu/

```
sudo docker ps -a # View all Docker containers in your system.
# For each container of our Docker image, run the following command:
sudo docker rm CONTAINER_ID_OR_NAME # Remove a Docker container.
# After removing all containers of our image, run the following command:
sudo docker image rm experiments # Delete our Docker image from your system.
```

Received 19 December 2024; accepted 12 August 2025