Composite Event Recognition with Arbitrary Specifications

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

[a] *NCSR "Demokritos", Greece*
[b] *University of Piraeus, Greece*

Abstract

Composite event recognition (CER) frameworks reason over streams of low-level, symbolic events in order to detect instances of spatio-temporal patterns defining high-level, composite activities. The Event Calculus is a temporal, logical formalism that has been used to define composite activities in CER, while RTEC$_\circ$ is a formal CER framework that detects composite activities based on their Event Calculus definitions. RTEC$_\circ$, however, cannot handle arbitrary Event Calculus definitions for composite activities, limiting the range of CER applications supported by RTEC$_\circ$. We propose RTEC$_{ft}$, an extension of RTEC$_\circ$ that supports arbitrary composite activity specifications in the Event Calculus. We present the syntax, semantics, reasoning algorithms and time complexity of RTEC$_{ft}$. Moreover, we propose a compiler for RTEC$_{ft}$, generating the optimal representation of an input set of Event Calculus definitions. We demonstrate the correctness of our compiler and outline its time complexity. We conducted an empirical evaluation of RTEC$_{ft}$ on synthetic and real data streams from human activity recognition and maritime situational awareness, including a comparison with two state-of-the-art Event Calculus-based systems, which demonstrates the benefits of RTEC$_{ft}$.

Keywords: Event Calculus, temporal pattern matching, composite event recognition

*Corresponding author.
Email addresses: pmantenoglou@iit.demokritos.gr (Periklis Mantenoglou), a.artikis@unipi.gr (Alexander Artikis)

1. Introduction

Composite event recognition (CER) involves the detection of composite activities by reasoning over streams of time-stamped, symbolic events [26, 32]. A CER framework employs a specification language, where it is possible to express the spatio-temporal combinations of input events that form each activity of interest in some application domain. In human activity recognition, e.g., we may specify the time periods during which two people are 'gathering' using a pattern stating that at least one of the two people is walking towards the other one, while, at the same time, the distance between them is a few meters and they are facing each other. As another example, in the task of monitoring composite maritime activities [8], we may define 'trawling', i.e., a type of fishing activity that involves several consecutive turns, as a sequence of 'change in heading' events.

The Event Calculus is a logic programming formalism for representing and reasoning about events and their effects over time [42]. It may be used as an activity specification language for CER, as it exhibits several desirable features, such as a formal, declarative semantics, and a support for relational and hierarchical activity specifications that may include background knowledge [51, 32]. We focus on the Run-Time Event Calculus (RTEC), which extends the Event Calculus with optimisation techniques for CER, such as windowing, indexing and caching [7]. In order to perform CER with minimal latency, RTEC processes hierarchies of composite activity definitions bottom-up, while caching and reusing the derived instances of composite activities, thus avoiding re-computations.

Unfortunately, RTEC does not support arbitrary composite activity definitions expressed in the Event Calculus. In human activity recognition, e.g., we may need to model activities defined in terms of the relative movement between persons P_1 and P_2, i.e., the concept '$movement(P_1, P_2)$'. For instance, expression '$movement(P_1, P_2) = gathering$' denotes that P_1 and P_2 are moving towards one another in order to have a meeting, while we may use expression '$movement(P_1, P_2) = abrupt\_gestures$' to denotes that, while P_1 and P_2 are talking to each other, one of them is moving his arms abruptly. Furthermore, it may be desirable to express that P_1 and P_2 may start making abrupt gestures to each other only after they have gathered close to one another, i.e., expression $movement(P_1, P_2) = abrupt\_gestures$ depends on expression $movement(P_1, P_2) = gathering$. RTEC does not support Event Calculus definitions where composite activities characterised by the same

underlying concept, such as $movement(P_1, P_2)$, depend on each other. To address this issue, we propose an extension of RTEC that supports arbitrary Event Calculus definitions.

Our starting point is $\text{RTEC}_\circ$, a version of RTEC that supports Event Calculus definitions with cyclic dependencies, which are often required for CER [48]. We propose $\text{RTEC}_{\not\!\circ}$, an extension of $\text{RTEC}_\circ$ that supports arbitrary composite activity definitions in the Event Calculus. This work is an extension of an earlier paper [46], where we presented the syntax and the semantics of $\text{RTEC}_{\not\!\circ}$, and proposed a compiler for $\text{RTEC}_{\not\!\circ}$ that converts an arbitrary set of composite activity definitions in the Event Calculus into a specification that guarantees correct run-time reasoning. The contributions of this work may be summarised as follows:

1. We demonstrate that a set of composite activity definitions generated by our compiler is optimal, i.e., $\text{RTEC}_{\not\!\circ}$ does not perform any redundant computations when reasoning with such a set of definitions.
2. We outline the time complexity of our compiler, which is linear to the number of composite activities and the number of conditions in the input set of composite activity definitions.
3. We present an empirical evaluation of $\text{RTEC}_{\not\!\circ}$ on synthetic and real data streams from human activity recognition and maritime situational awareness, including a comparison with two state-of-the-art frameworks that are based on the Event Calculus. The empirical analysis shows that $\text{RTEC}_{\not\!\circ}$ outperforms the state-of-the-art by several orders of magnitude, while being scalable over large-scale real-world event streams.

$\text{RTEC}_{\not\!\circ}$ and its compiler are publicly available[1].

2. Related Work

The literature contains numerous CER frameworks [2, 32], several of which are automata-based [56, 69, 33]. CORE, e.g., is a formal automata-based CER system that has proven to be more efficient than other contemporary automata-based engines [19]. CORE is restricted to unary relations, while the composite activities derived by CORE cannot be used as building blocks in other patterns. As a result, contrary to $\text{RTEC}_{\not\!\circ}$, CORE does not

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

support relational composite activities, hierarchical activity definitions and background knowledge.

There are CER approaches that extend SQL for handling data streams [4, 10, 39, 9, 68, 62]. StreamMill [43], e.g., is an SQL-based system that is restricted to non-blocking queries, i.e., queries that can be answered without having to wait for an arbitrarily long time for new event arrivals [67]. Although SQL-based formalisms are quite expressive, the queries required to specify CER patterns are commonly highly complicated, making them hard to understand and optimise [1].

There are also logic-based CER formalisms [27, 20, 16]. For instance, there are several frameworks supporting fragments of the LARS language [13] that are suitable for CER [14, 12, 28]. MeTeoR is a logic-based CER engine whose language extends DatalogMTL with windowing [64, 65]. The Chronicle Recognition System (CRS) represents composite activities as sets of events that are associated with time constraints [27]. The language of CRS includes several operators, such as sequencing, iteration and negation. TESLA provides a logic-based event specification language that supports negation, aggregates, iteration and event hierarchies, but does not support background knowledge [25]. The benefits of RTEC$_{fl}$ compared to the aforementioned approaches stem from its use of the Event Calculus, as it includes a built-in representation of inertia, allowing for succinct composite activity patterns, and thus supporting code maintenance and reasoning efficiency. At the same time, Event Calculus specifications are intuitive, paving the way for the cooperation between data scientist and domain (e.g., maritime) expert, as well as result explainability.

The Event Calculus has been employed in various settings, including mobility assistance [17], reactive and proactive health monitoring [22, 36] and simulations with cognitive agents [58]. The 'Macro Event Calculus', e.g., uses 'macro-events' to support composite event operators, such as sequence, disjunction, parallelism and iteration [21]. The 'Interval-based Event Calculus' incorporates durative events and supports sequencing, concurrency and negation [52]. The F2LP system maps Event Calculus rules into answer set programs, allowing the use of answer set programming solvers for reasoning [44]. Srinivasan et al. proposed an Event Calculus dialect with an integrated domain specification for biological feedback systems [60]. s(CASP) is a query-driven execution model for ASP with constraints, supporting Event Calculus-based reasoning [5].

The aforementioned Event Calculus-based approaches are not designed

for temporal pattern matching over streams of events for CER. Fusemate is a logic programming framework that integrates the Event Calculus with a description logic in order to reason over event streams [11]. jREC is a reactive implementation of the Cached Event Calculus [23] which is optimised for CER [15, 29]. Our work is based on RTEC, which performs CER with minimal latency; RTEC processes hierarchies of composite activity definitions bottom-up, while caching and reusing the derived instances of composite activities, thus avoiding re-computations. RTEC has proven highly efficient in demanding CER applications, including city transport management [7], maritime situational awareness [54] and commercial fleet management [63], outperforming the state-of-the-art [48, 47, 63]. In Section 7, we compare RTEC_{fl} with Fusemate and the most recent implementation of jREC [29].

One of the key advantages of RTEC is its interval-based semantics. Composite activities are typically durative, and thus need to be represented using temporal intervals [32]. Using sets of individual time-points to represent occurrences of durative situations fails to capture ongoing activities and leads to semantic ambiguities in cases of compositional/hierarchical patterns [30, 50, 66].

D^2IA is a CER framework that augments the Big Data engine Flink with interval-based semantics [9]. The language of D^2IA extends CQL [4] with a set of temporal operators for reasoning over durative events. Compared to RTEC, the language of D^2IA has several limitations. In D^2IA, e.g., it is not straightforward to combine instantaneous and durative events in the conditions of a pattern, while negation is limited to the absence of a specified type of instantaneous event between the occurrences of two other events. Moreover, D^2IA does not support background knowledge in event patterns. TPStream [39] is a CER framework that performs temporal pattern matching over intervals of durative activities, which are derived based on instantaneous input events. In TPStream, durative activities cannot be defined in terms of multiple event types or background knowledge. ISEQ [45] performs temporal pattern matching over streams of durative activities. ISEQ does not allow for the derivation of durative activities from instantaneous events. Unlike RTEC, neither ISEQ nor TPStream supports relational patterns, which is a significant limitation for CER. ETALIS [3] is a CER system that supports interval-based reasoning with background knowledge. ETALIS does not allow for the explicit representation of time, complicating the specification of fluent value changes, including the formalisation of the common-sense law of inertia, which is innate in the Event Calculus.

Table 1: Main predicates of RTEC$_\circ$.

Predicate	Meaning
$\mathsf{happensAt}(E, T)$	Event E occurs at time T.
$\mathsf{initiatedAt}(F = V, T)$	At time T, a period of time during which fluent F has value V is initiated.
$\mathsf{terminatedAt}(F = V, T)$	At time T, a period of time during which fluent F has value V is terminated.
$\mathsf{holdsAt}(F = V, T)$	The value of fluent F is V at time T.
$\mathsf{holdsFor}(F = V, I)$	Fluent F has value V continuously in the maximal intervals included in list I.

3. Background

Our starting point is RTEC$_\circ$, i.e., a recent extension of the Run-Time Event Calculus (RTEC) that supports efficient reasoning over temporal specifications with cyclic dependencies [48]. We present the syntax, semantics and reasoning algorithms of RTEC$_\circ$. In Section 4, we outline the limitations of RTEC$_\circ$, and, in Section 5, we present an extension of RTEC$_\circ$ that supports arbitrary Event Calculus definitions.

3.1. Syntax & Semantics

The language of RTEC$_\circ$ follows the Event Calculus, which is many-sorted, including sorts for representing time, instantaneous events and 'fluents', i.e., properties that may have different values at different points in time. The time model comprises a linear time-line with non-negative integer time-points. $\mathsf{happensAt}(E, T)$ signifies that event E occurs at time-point T. $\mathsf{initiatedAt}(F = V, T)$ (resp. $\mathsf{terminatedAt}(F = V, T)$) expresses that a time period during which a fluent F has the value V continuously is initiated (terminated) at time-point T. $\mathsf{holdsAt}(F = V, T)$ states that F has value V at T, while $\mathsf{holdsFor}(F = V, I)$ expresses that the 'fluent-value pair' (FVP) $F{=}V$ holds continuously in the maximal intervals included in list I. Table 1 summarises these main predicates of RTEC$_\circ$.

In CER, $\mathsf{happensAt}$ is used to express the input events of the stream, while FVPs express composite activities. A formalisation of all activities of a domain of interest in the Event Calculus is called *event description*.

Definition 1 (Event Description)**.** An event description $\mathcal{E}$ is a set of:

- ground $\mathsf{happensAt}(E, T)$ facts, expressing a stream of event instances, and
- rules with head $\mathsf{initiatedAt}(F = V, T)$ or $\mathsf{terminatedAt}(F = V, T)$, expressing the effects of events on FVP $F{=}V$. ■

Definition 2 (Syntax of the Rules in the Event Description)**.** Rules with head $\mathsf{initiatedAt}(F = V, T)$ have the following syntax:

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

The first body literal of an $\mathsf{initiatedAt}$ rule is a positive $\mathsf{happensAt}$ predicate; this is followed by a possibly empty set, denoted by '[[ ]]', of positive/negative $\mathsf{happensAt}$ and $\mathsf{holdsAt}$ predicates. 'not' expresses negation-by-failure [24], while '[not]' denotes that 'not' is optional. All (head and body) predicates are evaluated on the same time-point T. The bodies of $\mathsf{terminatedAt}(F = V, T)$ rules have the same form. ■

Example 1 (Event Description for Human Activity Recognition)**.** In human activity recognition, we apply rules on streams containing symbolic representations of video feeds [6]. In general, such rules are constructed in collaboration with domain experts or learned from data [38]. We use the fluent $\mathit{interaction}(P_1, P_2)$ to express that people P_1 and P_2 are interacting, while the value of $\mathit{interaction}(P_1, P_2)$ denotes the stage of the interaction. The '*greeting*' stage of $\mathit{interaction}(P_1, P_2)$ denotes that P_1 and P_2 are greeting each other at a distance. Below, we outline a set of rules comprising the

specification of FVP $interaction(P_1, P_2) = greeting$:

$$\begin{aligned}
&\mathsf{initiatedAt}(interaction(P_1, P_2) = greeting,\ T) \leftarrow \\
&\quad \mathsf{happensAt}(active(P_1),\ T), \\
&\quad \mathsf{happensAt}(active(P_2),\ T), \\
&\quad \mathsf{holdsAt}(distance(P_1, P_2) = mid,\ T), \\
&\quad \mathsf{holdsAt}(orientation(P_1, P_2) = facing,\ T).
\end{aligned} \tag{2}$$

$$\begin{aligned}
&\mathsf{terminatedAt}(interaction(P_1, P_2) = greeting,\ T) \leftarrow \\
&\quad \mathsf{happensAt}(walking(P_1),\ T), \\
&\quad \mathsf{not\ holdsAt}(orientation(P_1, P_2) = facing,\ T).
\end{aligned} \tag{3}$$

$$\begin{aligned}
&\mathsf{terminatedAt}(interaction(P_1, P_2) = greeting,\ T) \leftarrow \\
&\quad \mathsf{happensAt}(walking(P_2),\ T), \\
&\quad \mathsf{not\ holdsAt}(orientation(P_1, P_2) = facing,\ T).
\end{aligned} \tag{4}$$

According to rule (2), P_1 and P_2 start greeting when both of them are 'active', i.e., moving their arms while in the same position, the distance between them is a few meters, denoted by the value 'mid', and they are facing towards one another. Rules (3)–(4) express that P_1 and P_2 stop greeting when one of them starts walking, while they are not facing each other. The FVPs $distance(P_1, P_2) = mid$ and $orientation(P_1, P_2) = facing$ are defined based on the coordinates and the orientation of the tracked people, which are provided in the input stream.

Moreover, we may use the fluent $movement(P_1, P_2)$ to express the relative movement between people P_1 and P_2 and the value '*gathering*' of $movement(P_1, P_2)$ to denote that P_1 and P_2 are approaching one another. The specification of FVP $movement(P_1, P_2) = gathering$ comprises the fol-

lowing rules:

$$\begin{aligned}&\mathsf{initiatedAt}(movement(P_1, P_2) = gathering,\ T) \leftarrow \\ &\quad \mathsf{happensAt}(walking(P_1),\ T), \\ &\quad \mathsf{holdsAt}(distance(P_1, P_2) = mid,\ T), \\ &\quad \mathsf{holdsAt}(orientation(P_1, P_2) = facing,\ T).\end{aligned} \tag{5}$$

$$\begin{aligned}&\mathsf{initiatedAt}(movement(P_1, P_2) = gathering,\ T) \leftarrow \\ &\quad \mathsf{happensAt}(walking(P_2),\ T), \\ &\quad \mathsf{holdsAt}(distance(P_1, P_2) = mid,\ T), \\ &\quad \mathsf{holdsAt}(orientation(P_1, P_2) = facing,\ T).\end{aligned} \tag{6}$$

$$\begin{aligned}&\mathsf{terminatedAt}(movement(P_1, P_2) = gathering,\ T) \leftarrow \\ &\quad \mathsf{happensAt}(active(P_1),\ T), \\ &\quad \mathsf{not\ happensAt}(walking(P_2),\ T).\end{aligned} \tag{7}$$

$$\begin{aligned}&\mathsf{terminatedAt}(movement(P_1, P_2) = gathering,\ T) \leftarrow \\ &\quad \mathsf{happensAt}(active(P_2),\ T), \\ &\quad \mathsf{not\ happensAt}(walking(P_1),\ T).\end{aligned} \tag{8}$$

Rules (5)–(6) state that P_1 and P_2 start gathering when one of them is walking towards the other person, while their distance is a few meters and they are facing each other. Rules (7)–(8) express that P_1 and P_2 stop gathering when one of them is being active, while the other person is not walking. $\Diamond$

The dependencies among the FVPs in an event description can be expressed in the form of a *dependency graph*.

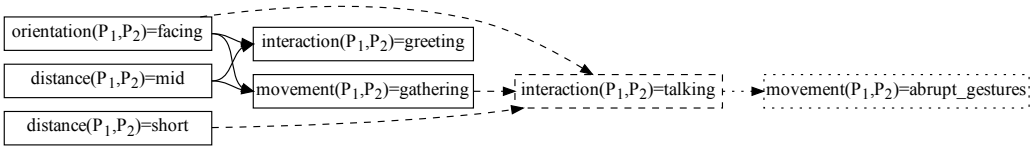
Definition 3 (Dependency Graph)**.** The dependency graph of an event description is a directed graph $G = (\mathcal{V}, \mathcal{E})$, where:

1. $\mathcal{V}$ contains one vertex $v_{F=V}$ for each FVP $F = V$.
2. $\mathcal{E}$ contains an edge $(v_{F_j=V_j}, v_{F_i=V_i})$ iff there is an $\mathsf{initiatedAt}/\mathsf{terminatedAt}$ rule for $F_i = V_i$ having $\mathsf{holdsAt}(F_j = V_j, T)$ as one of its conditions. ■

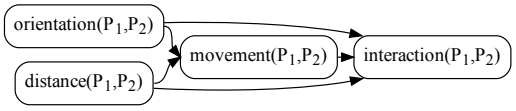
The vertices and edges of Figure 1a that are drawn with continuous lines, e.g., comprise the dependency graph $G_{\mathcal{E}_1}$ of event description $\mathcal{E}_1$, which contains rules (2)–(8) of Example 1.

Based on the dependency graph of an event description, it is possible to define a function *level* that maps the FVPs of the event description to the positive integers. Towards defining an FVP level function, we define the *level of a vertex* in a directed acyclic graph as follows:

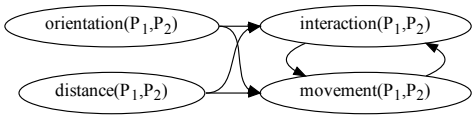
Definition 4 (Vertex Level)**.** Given a directed acyclic graph, the level of a vertex v is equal to:



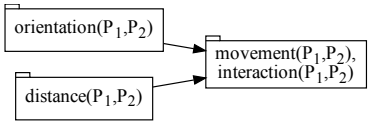
(a) The dependency graph $G_{\mathcal{E}_1}$ of event description $\mathcal{E}_1$ (continuous lines), the dependency graph $G_{\mathcal{E}_2}$ of event description $\mathcal{E}_2$ (continuous and dashed lines), and the dependency graph $G_{\mathcal{E}_3}$ of event description $\mathcal{E}_3$ (all lines). For simplicity, a vertex v_j is displayed as j.



(b) The fluent dependency graph $G^{fl}_{\mathcal{E}_2}$ of $\mathcal{E}_2$. The contracted fluent dependency graph $G^{cdfl}_{\mathcal{E}_2}$ of $\mathcal{E}_2$ is the same as $G^{fl}_{\mathcal{E}_2}$.



(c) The fluent dependency graph $G^{fl}_{\mathcal{E}_3}$ of $\mathcal{E}_3$.



(d) The contracted fluent dependency graph $G^{cdfl}_{\mathcal{E}_3}$ of $\mathcal{E}_3$. We display a vertex v_{S_i} of a contracted fluent dependency graph, where S_i is a SCC of the corresponding fluent dependency graph, as the set of fluents whose vertices are in S_i.

Figure 1: Dependency graphs, fluent dependency graphs and contracted fluent dependency graphs. We use distinct shapes for the vertices of each type of graph to aid the presentation.

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 $n-1$, and zero or more incoming edges from vertices of levels lower than $n-1$. ■

A dependency graph may or may not be acyclic. Given an acyclic dependency graph, the level of an FVP $F=V$ is defined as the level of vertex $v_{F=V}$ in the dependency graph. In the acyclic dependency graph of Figure 1a, e.g., $v_{interaction(P_1,P_2)=greeting}$ has level *2*, and thus $interaction(P_1, P_2) = greeting$ has level *2*. In order to handle cyclic dependency graphs, we employ the *contracted dependency graph* of an event description, which is, by definition, acyclic. Then, we define the *level of an FVP* based on the level of the corresponding vertex in the contracted dependency graph.

A directed graph becomes acyclic by contracting its strongly connected

components (SCC)s into single vertices.

Definition 5 (SCC Contracted Graph)**.** Given a directed graph $G=(\mathcal{V},\mathcal{E})$ and the SCCs $S_1, S_2, \ldots, S_n$ of G, the SCC contracted graph $G^{cd}=(\mathcal{V}^{cd},\mathcal{E}^{cd})$ of G is defined as follows:

1. $\mathcal{V}^{cd}=\bigcup_{1\leq i\leq n}\{v_{S_i}\}$.
2. $(v_{S_i},\ v_{S_j})\in\mathcal{E}^{cd}$ iff $\exists v_i, v_j\in\mathcal{V}$, such that $v_i\in S_i$, $v_j\in S_j$, $S_i\neq S_j$ and $(v_i,\ v_j)\in\mathcal{E}$. ■

Definition 6 (Contracted Dependency Graph)**.** Consider an event description with dependency graph G. The contracted dependency graph of the event description is the SCC contracted graph of G. ■

The dependency graph $G_{\mathcal{E}_1}$ in Figure 1a is acyclic, i.e., every SCCs of $G_{\mathcal{E}_1}$ contains one vertex. As a result, the contracted dependency graph $G^{cd}_{\mathcal{E}_1}$ of $G_{\mathcal{E}_1}$ is the same as $G_{\mathcal{E}_1}$.

Definition 7 (FVP Level in RTEC$_\circ$)**.** Consider an event description with dependency graph G and contracted dependency graph G^{cd}. The level of an FVP $F=V$, such that vertex $v_{F=V}$ is included in SCC S_i of G, is equal to the level of vertex v_{S_i} in G^{cd}. ■

RTEC$_\circ$ supports event descriptions where FVPs with the same fluent have the same FVP level. For such an event description, a local stratification may be constructed as follows. The first stratum contains all groundings of happensAt. The remaining strata are formed by following, in a bottom-up fashion, the levels of FVPs. For each FVP level l without cyclic dependencies, we have one stratum containing the ground predicates for FVPs with level l. For each FVP level l with cyclic dependencies, the ground predicates for FVPs with level l have to be stratified further in terms of their time-stamp. We introduce an additional stratum for each time-point of the window, i.e., the finite portion of the stream currently being processing by RTEC$_\circ$.

Proposition 1 (Semantics of RTEC$_\circ$)**.** Consider an event description $\mathcal{E}$ where the FVPs with the same fluent have the same FVP level (see Definition 7). $\mathcal{E}$ is a locally stratified logic program [57]. ♦

3.2. Reasoning

The key reasoning task of RTEC$_\circ$ is the computation of holdsFor$(F=V, I)$, i.e., the list of maximal intervals I during which each FVP $F=V$ of the event description holds continuously. Recall that, in CER, FVPs express the composite activities that we are interested in detecting. RTEC$_\circ$ computes list I

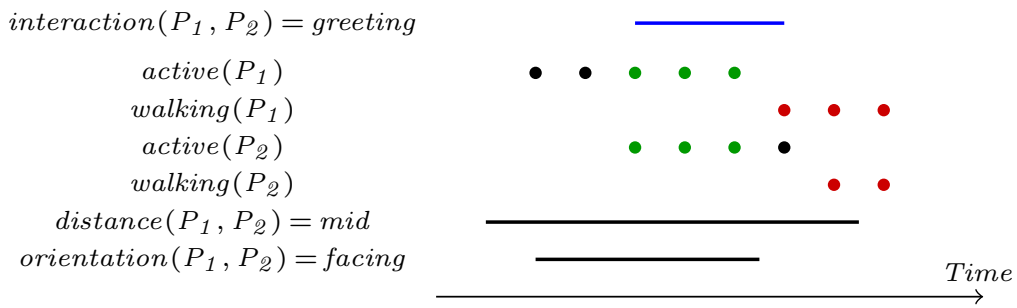


Figure 2: Maximal interval computation for $interaction(P_1, P_2) = greeting$. Each row of dots (resp. horizontal lines) denotes the instantaneous occurrences (maximal intervals) of the event (FVP) reported at the start of the row. Green (red) dots signify events leading to an initiation (termination) of $interaction(P_1, P_2) = greeting$. Black (blue) lines denote input (output) maximal intervals.

in $\mathsf{holdsFor}(F = V, I)$ as follows. First, it computes the initiations of $F = V$ based on the rules of the event description with head $\mathsf{initiatedAt}(F = V, T)$. Second, if there is at least one initiation of $F = V$, then RTEC$_\circ$ computes the terminations of $F = V$ based on the rules with head $\mathsf{terminatedAt}(F = V, T)$, as well as the rules with head $\mathsf{initiatedAt}(F = V', T)$, where $V' \neq V$. Third, RTEC$_\circ$ computes the maximal intervals of $F = V$ by matching each initiation T_s of $F = V$ with the first termination T_e of $F = V$ after T_s, ignoring every intermediate initiation between T_s and T_e. $\mathsf{holdsAt}(F = V, T)$ may then be evaluated by checking whether T belongs to one of the maximal intervals of FVP $F = V$.

Figure 2 presents an example for the computation of holdsFor, where RTEC$_\circ$ evaluates $\mathsf{holdsFor}(interaction(P_1, P_2) = greeting, I)$ using rules (2)–(4). First, RTEC$_\circ$ computes the initiations of $interaction(P_1, P_2) = greeting$. Based on rule (2), there is an initiation of $interaction(P_1, P_2) = greeting$ at each time-point where both P_1 and P_2 are 'active', i.e., events $active(P_1)$ and $active(P_2)$ take place, while they are facing each other at a distance of a few meters, i.e., FVPs $orientation(P_1, P_2) = facing$ and $distance(P_1, P_2) = mid$ hold. Based on the input in Figure 2, these conditions are satisfied at three time-points—denoted by coloring the corresponding events green—which comprise the initiations of $interaction(P_1, P_2) = greeting$. Afterwards, RTEC$_\circ$ computes the terminations of $interaction(P_1, P_2) = greeting$. According to rules (3)–(4), there is a termination of $interaction(P_1, P_2) = greeting$ at each time-point where P_1 and P_2 are not facing each other, i.e., FVP

orientation$(P_1, P_2) =$ *facing* does not hold, and one of them starts walking, i.e., event *walking*(P_1) or event *walking*(P_2) takes place. Thus, based on our input, there are three terminations of *interaction*$(P_1, P_2) =$ *greeting*, which were the result of the events colored red. Having computed the initations and the terminations of *interaction*$(P_1, P_2) =$ *greeting*, RTEC$_\circ$ proceeds to the computation of its maximal intervals. In our example, there is one maximal interval for *interaction*$(P_1, P_2) =$ *greeting*, constructed by matching its first initiation with its first termination, ignoring the two intermediate initiations (see the interval colored blue in Figure 2).

RTEC$_\circ$ processes FVPs in a bottom-up manner, computing and caching their intervals level-by-level. In order to derive the initiations and the terminations of an FVP $F=V$, we evaluate the initiatedAt and terminatedAt rules defining $F=V$. The body of such a rule may include a holdsAt$(F'=V', T)$ condition (see rule schema (1)), leading to an edge $(v_{F'=V'}, v_{F=V})$ in the dependency graph (see Definition 3). We distinguish two cases for the evaluation of holdsAt$(F'=V', T)$:

1. Vertices $v_{F'=V'}$ and $v_{F=V}$ are not part of a cycle in the dependency graph. In this case, $v_{F'=V'}$ and $v_{F=V}$ are in different SCCs of the dependency graph and, based on edge $(v_{F'=V'}, v_{F=V})$, $F'=V'$ has a lower level than $F=V$ (see Definition 7). Since RTEC$_\circ$ processes FVPs in ascending FVP level order, at the time of processing $F=V$, the intervals of $F'=V'$ that are required to compute holdsAt$(F'=V', T)$ have been derived and cached at a previous step. Thus, holdsAt$(F'=V', T)$ is resolved by fetching the intervals of $F'=V'$ from the cache and checking whether T belongs to one of those intervals, without the need for re-computation.
2. Vertices $v_{F'=V'}$ and $v_{F=V}$ are part of a cycle in the dependency graph. In this case, $v_{F'=V'}$ and $v_{F=V}$ are in the same SCC of the dependency graph, and thus $F'=V'$ and $F=V$ have the same level (see Definition 7). As a result, RTEC$_\circ$ may process $F=V$ before $F'=V'$, in which case the intervals of $F'=V'$ are not be present in the cache at the time of processing $F=V$. To address this issue, RTEC$_\circ$ computes holdsAt$(F'=V', T)$ using the incremental caching techniques presented in [48].

4. Problem Statement

Towards a more accurate domain specification for human activity recognition, we may extend event description $\mathcal{E}_1$ of Example 1 with a definition for an FVP expressing that two people are talking.

Example 2 (Representing $interaction(P_1, P_2) = talking$ (Example 1 cont'd))**.** After having approached one another, persons P_1 and P_2 may start talking, in which case the value of the $interaction(P_1, P_2)$ fluent should change from '*greeting*' to '*talking*'. The specification of FVP $interaction(P_1, P_2) = talking$ comprises the following rules:

$$\begin{aligned}
&\mathsf{initiatedAt}(interaction(P_1, P_2) = talking,\ T) \leftarrow \\
&\quad \mathsf{happensAt}(active(P_1),\ T), \\
&\quad \mathsf{holdsAt}(distance(P_1, P_2) = short,\ T), \\
&\quad \mathsf{holdsAt}(orientation(P_1, P_2) = facing,\ T), \\
&\quad \mathsf{not\ holdsAt}(movement(P_1, P_2) = gathering,\ T).
\end{aligned} \tag{9}$$

$$\begin{aligned}
&\mathsf{initiatedAt}(interaction(P_1, P_2) = talking,\ T) \leftarrow \\
&\quad \mathsf{happensAt}(active(P_2),\ T), \\
&\quad \mathsf{holdsAt}(distance(P_1, P_2) = short,\ T), \\
&\quad \mathsf{holdsAt}(orientation(P_1, P_2) = facing,\ T), \\
&\quad \mathsf{not\ holdsAt}(movement(P_1, P_2) = gathering,\ T).
\end{aligned} \tag{10}$$

$$\begin{aligned}
&\mathsf{terminatedAt}(interaction(P_1, P_2) = talking,\ T) \leftarrow \\
&\quad \mathsf{happensAt}(inactive(P_1),\ T), \\
&\quad \mathsf{happensAt}(inactive(P_2),\ T).
\end{aligned} \tag{11}$$

According to rules (9)–(10), P_1 and P_2 start talking when one of them is being active, while their distance is about one meter, denoted by 'short', they are facing one another and their relative movement is not 'gathering', i.e., P_1 and P_2 are not moving towards one another. Rule (11) denotes that P_1 and P_2 stop talking when neither of them is being active. ◊

A fluent cannot have more than one value at any time; an initiation of an FVP $F = V_1$ implies a termination of FVP $F = V_2$, where $V_1 \neq V_2$. As a result, there are implicit dependencies among FVPs with the same fluent. In the event description of Example 2, e.g., FVPs $interaction(P_1, P_2) = greeting$ and $interaction(P_1, P_2) = talking$ implicitly depend on each other.

The vertices and edges of Figure 1a that are drawn with continuous or dashed lines comprise the dependency graph $G_{\mathcal{E}_2}$ of event description $\mathcal{E}_2$, i.e., the extension of event description $\mathcal{E}_1$ with rules (9)–(11) of Example 2. FVPs

interaction$(P_1, P_2) =$ *greeting* and *movement*$(P_1, P_2) =$ *gathering* have level *2*, while FVP *interaction*$(P_1, P_2) =$ *talking* has level *3* (see Definition 7).

Event description $\mathcal{E}_2$ contains FVPs with the same fluent and different levels, which is common in CER specifications. In city transport management, e.g., fluent '*punctuality*(Vh)' may be used to monitor the punctuality level of a vehicle Vh over time [7]. *punctuality*$(Vh) =$ *low* may be initiated when Vh leaves a stop earlier than scheduled while *punctuality*$(Vh) =$ *mid* holds. As another example, in maritime activity monitoring, we may employ the fluent '*fishing_trip*(Vl)' to survey a fishing trip of a vessel Vl [54]. FVP *fishing_trip*$(Vl) =$ *ended* may depend on FVP *fishing_trip*$(Vl) =$ *returning*, which expresses the previous stage of the trip. In these cases, the level of FVP *punctuality*$(Vh) =$ *low* is higher than the level of *punctuality* $=$ *mid*, while *fishing_trip*$(Vl) =$ *ended* has a higher level than *fishing_trip*$(Vl) =$ *returning* (see Definition 7).

RTEC$_\circ$ does not support event descriptions, such as $\mathcal{E}_2$, where FVPs with the same fluent have different levels. Suppose that FVP $F = V_1$ has level n and FVP $F = V_2$ has level m, where $n < m$, and that RTEC$_\circ$ is currently processing the FVPs with level n. When processing $F = V_1$, RTEC$_\circ$ needs to evaluate the rules with head $\mathsf{initiatedAt}(F = V_2, T)$, as the initiation of $F = V_2$ constitute terminations of $F = V_1$. Such a rule may include a body condition referring to an FVP $F' = V'$ with level n', where $n \leq n' < m$. Since $F' = V'$ has a lower level than $F = V_2$, RTEC$_\circ$ attempts to evaluate $\mathsf{holdsAt}(F' = V', T)$ by retrieving the intervals of $F' = V'$ from the cache, in order to check whether T belongs to one of them. However, the cache of RTEC$_\circ$ may not contain the intervals of $F' = V'$ at this time, because $F' = V'$ has level n' and RTEC$_\circ$ is currently processing the FVPs with level n, where $n \leq n'$, compromising correctness.

In the case of $\mathcal{E}_2$, when processing *interaction*$(P_1, P_2) =$ *greeting*, RTEC$_\circ$ evaluates the initiations of *interaction*$(P_1, P_2) =$ *talking*, as they are terminations of *interaction*$(P_1, P_2) =$ *greeting*. According to rules (9)–(10) of event description $\mathcal{E}_2$, the initiations of *interaction*$(P_1, P_2) =$ *talking* depend on FVP *movement*$(P_1, P_2) =$ *gathering*, whose intervals may not present in the cache at the time of processing *interaction*$(P_1, P_2) =$ *greeting*. For this reason, RTEC$_\circ$ does not support event description $\mathcal{E}_2$.

We may address this issue by assigning to *interaction*$(P_1, P_2) =$ *greeting* a higher level than the level of *movement*$(P_1, P_2) =$ *gathering*. According to dependency graph $G_{\mathcal{E}_2}$ (see Figure 1a), since there is no FVP that depends on FVP *interaction*$(P_1, P_2) =$ *greeting*, we may increase the level of

$interaction(P_1, P_2) = greeting$ to 3 without producing an FVP level assignment that compromises the correctness of the bottom-up processing of RTEC$_\circ$. In this way, FVP $movement(P_1, P_2) = gathering$ is processed by RTEC$_\circ$ before FVP $interaction(P_1, P_2) = greeting$, and thus, at the time of processing $interaction(P_1, P_2) = greeting$, the cache of RTEC$_\circ$ contains the maximal intervals of $movement(P_1, P_2) = gathering$, avoiding the aforementioned error.

However, it is not always possible to circumvent the issues introduced by FVPs with the same fluent and different levels by increasing the level of an FVP. Consider the following example, where we extend event description $\mathcal{E}_2$ with a definition for an FVP expressing that two people are making abrupt movements while talking.

Example 3 (Representing $movement(P_1, P_2) = abrupt\_gestures$ (Example 2 cont'd))**.** While people P_1 and P_2 are talking, they may start moving their arms abruptly, possibly indicating that a fight between P_1 and P_2 is about to start. The specification of FVP $movement(P_1, P_2) = abrupt\_gestures$ comprises the following rules:

$$\begin{aligned}
&\mathsf{initiatedAt}(movement(P_1, P_2) = abrupt_gestures,\ T) \leftarrow \\
&\quad \mathsf{happensAt}(abrupt(P_1),\ T), \\
&\quad \mathsf{holdsAt}(interaction(P_1, P_2) = talking,\ T).
\end{aligned} \tag{12}$$

$$\begin{aligned}
&\mathsf{initiatedAt}(movement(P_1, P_2) = abrupt_gestures,\ T) \leftarrow \\
&\quad \mathsf{happensAt}(abrupt(P_2),\ T), \\
&\quad \mathsf{holdsAt}(interaction(P_1, P_2) = talking,\ T).
\end{aligned} \tag{13}$$

$$\begin{aligned}
&\mathsf{terminatedAt}(movement(P_1, P_2) = abrupt_gestures,\ T) \leftarrow \\
&\quad \mathsf{happensAt}(active(P_1),\ T), \\
&\quad \mathsf{not\ happensAt}(abrupt(P_2),\ T).
\end{aligned} \tag{14}$$

$$\begin{aligned}
&\mathsf{terminatedAt}(movement(P_1, P_2) = abrupt_gestures,\ T) \leftarrow \\
&\quad \mathsf{happensAt}(active(P_2),\ T), \\
&\quad \mathsf{not\ happensAt}(abrupt(P_1),\ T).
\end{aligned} \tag{15}$$

Rules (12)–(13) denote that $movement(P_1, P_2) = abrupt\_gestures$ is initiated when one of the people P_1 and P_2 starts moving abruptly while the two of them are talking. Rules (14)–(15) express that we have a termination of $movement(P_1, P_2) = abrupt\_gestures$ when one of the two people starts being active while the other one is not moving abruptly. $\Diamond$

All the vertices and edges in Figure 1a compose dependency graph $G_{\mathcal{E}_3}$ of event description $\mathcal{E}_3$, i.e., the extension of event description $\mathcal{E}_2$ with rules

(12)–(15). According to $G_{\mathcal{E}_3}$, FVP $movement(P_1, P_2) = abrupt\_gestures$ has level *4*.

Event description $\mathcal{E}_3$ contains FVPs with the same fluent and different levels. FVPs $interaction(P_1, P_2) = greeting$ and $interaction(P_1, P_2) = talking$ have level *2* and *3*, respectively, while FVPs $movement(P_1, P_2) = gathering$ and $movement(P_1, P_2) = abrupt\_gestures$ have level *2* and *4*. As a result, RTEC$_\circ$ does not support event description $\mathcal{E}_3$. When processing FVP $movement(P_1, P_2) = gathering$, RTEC$_\circ$ may need to evaluate the terminations of $movement(P_1, P_2) = gathering$, which include the initiations of FVP $movement(P_1, P_2) = abrupt\_gestures$. According to rules (12)–(13), these initiations depend on $interaction(P_1, P_2) = talking$, whose intervals are not present in the cache at this time.

In this case, we cannot set the level of $movement(P_1, P_2) = gathering$ to *4*, with the goal of processing FVP $interaction(P_1, P_2) = talking$ before $movement(P_1, P_2) = gathering$, because dependency graph $G_{\mathcal{E}_3}$ contains edge $(v_{movement(P_1,P_2)=gathering}, v_{interaction(P_1,P_2)=talking})$, implying that we cannot compute the maximal intervals of $interaction(P_1, P_2) = talking$ before processing $movement(P_1, P_2) = gathering$. Thus, these FVPs should have the same level. Moreover, FVP $interaction(P_1, P_2) = greeting$ depends on FVP $interaction(P_1, P_2) = talking$, and vice versa, which means that these FVPs should also have the same level. Therefore, all the aforementioned FVPs should have the same level, i.e., *2*, implying that they must be processed with incremental caching (see the second case presented in Section 3.2).

5. Proposed Solution: RTEC$_{\not\!\!\circ}$

We propose RTEC$_{\not\!\!\circ}$, an extension of RTEC$_\circ$ that supports event descriptions where the vertices of FVPs with the same fluent may have different levels, such as event descriptions $\mathcal{E}_2$ and $\mathcal{E}_3$. To achieve this, RTEC$_{\not\!\!\circ}$ incorporates a new definition for FVP level that takes into account the implicit dependencies between FVPs with the same fluent. We demonstrate that, based on the definition of FVP level in RTEC$_{\not\!\!\circ}$, we may construct a local stratification for every possible event description. Afterwards, we propose a compiler for RTEC$_{\not\!\!\circ}$ that assigns a level to each FVP of an input event description and identifies the $\mathsf{holdsAt}(F = V, T)$ conditions that need to be resolved with the incremental caching technique proposed in [48], because the intervals of $F{=}V$ may not be present in the cache at the time of evaluating $\mathsf{holdsAt}(F = V, T)$. We prove the correctness of our compiler, demonstrat-

ing that it generates event descriptions that are optimal with respect to the representation of holdsAt conditions, and outline its time complexity. Moreover, we outline the cost of RTEC$_{fl}$, showing that it is the same as the cost of RTEC$_{\circ}$. Therefore, RTEC$_{fl}$ extends the range of temporal specifications supported by RTEC$_{\circ}$, while maintaining its high reasoning efficiency.

5.1. Syntax & Semantics

In RTEC$_{fl}$, all FVPs with the same fluent have the same level. This is achieved by determining FVP level based on the *fluent dependency graph* of the event description, which is defined as follows:

Definition 8 (Fluent Dependency Graph)**.** Consider an event description with dependency graph $G=(\mathcal{V},\mathcal{E})$. The fluent dependency graph of the event description is a directed graph $G^{fl}=(\mathcal{V}^{fl},\mathcal{E}^{fl})$, where:

1. $\mathcal{V}^{fl}$ contains one vertex v_F for each fluent F.
2. $\mathcal{E}^{fl}$ contains an edge (v_{F_1},v_{F_2}), where $F_1 \neq F_2$, iff there is an edge $(v_{F_1=V_1},v_{F_2=V_2})$ in $\mathcal{E}$, where V_1 and V_2 are values of fluents F_1 and F_2, respectively. ■

Figure 1b, e.g., depicts the fluent dependency graph $G^{fl}_{\mathcal{E}_2}$ of event description $\mathcal{E}_2$ of Example 2. Vertex $v_{interaction(P_1,P_2)}$ of $G^{fl}_{\mathcal{E}_2}$ corresponds to vertices $v_{interaction(P_1,P_2)=greeting}$ and $v_{interaction(P_1,P_2)=talking}$ of $G_{\mathcal{E}_2}$, inheriting their incoming edges.

The fluent dependency graph $G^{fl}_{\mathcal{E}_2}$ is acyclic. Therefore, we may assign to each FVP $F=V$ of event description $\mathcal{E}_2$ the level of vertex v_F in the fluent dependency graph $G^{fl}_{\mathcal{E}_2}$, which is derived by following Definition 4. It could be the case, however, that the fluent dependency graph of an event description contains cycles. Figure 1c, e.g., depicts the fluent dependency graph $G^{fl}_{\mathcal{E}_3}$ of event description $\mathcal{E}_3$. $G^{fl}_{\mathcal{E}_3}$ includes a cycle, while, according to Definition 4, the level of a vertex is defined only on acyclic graphs. To address this issue, we contract the vertices of the fluent dependency graph that are in the same strongly connected component (SCC), leading to an acyclic graph. We define the *contracted fluent dependency graph* as follows:

Definition 9 (Contracted Fluent Dependency Graph)**.** Consider an event description with fluent dependency graph G^{fl}. The contracted fluent dependency graph G^{cdfl} of the event description is the SCC contracted graph of G^{fl}. ■

Consider, e.g., the fluent dependency graph $G^{fl}_{\mathcal{E}_2}$ of Figure 1b. $G^{fl}_{\mathcal{E}_2}$ is acyclic, and thus every SCC of $G^{fl}_{\mathcal{E}_2}$ contains one vertex. As a result, the contracted fluent dependency graph $G^{cdfl}_{\mathcal{E}_2}$ of $G^{fl}_{\mathcal{E}_2}$ is the same as $G^{fl}_{\mathcal{E}_2}$. As another example, Figure 1d presents the contracted fluent dependency graph $G^{cdfl}_{\mathcal{E}_3}$ corresponding to the fluent dependency graph $G^{fl}_{\mathcal{E}_3}$ in Figure 1c, which is produced by contracting vertices $v_{movement(P_1,P_2)}$ and $v_{interaction(P_1,P_2)}$ of $G^{fl}_{\mathcal{E}_3}$, as these vertices are in the same SCC of $G^{fl}_{\mathcal{E}_3}$. Due to this contraction of vertices, $G^{cdfl}_{\mathcal{E}_3}$ is acyclic.

We may assign a level to each vertex in a contracted fluent dependency graph by following Definition 4. We define the *level of an FVP* in RTEC$_{fl}$ as follows:

Definition 10 (FVP Level in RTEC$_{fl}$)**.** Consider an event description with fluent dependency graph G^{fl} and contracted fluent dependency graph G^{cdfl}. The level of an FVP $F=V$, such that vertex v_F is included in SCC S_i of G^{fl}, is equal to the level of vertex v_{S_i} of G^{cdfl}. ■

Based on Definition 10, FVPs with the same fluent have the same level. In the case of event description $\mathcal{E}_2$, e.g., where the contracted fluent dependency graph $G^{cdfl}_{\mathcal{E}_2}$ of $\mathcal{E}_2$ matches with the fluent dependency graph in Figure 1b, FVPs $interaction(P_1, P_2) = greeting$ and $interaction(P_1, P_2) = talking$ have level *3* because the level of vertex $v_{interaction(P_1,P_2)}$ in $G^{cdfl}_{\mathcal{E}_2}$ is *3*. In the case of event description $\mathcal{E}_3$, the vertex of the contracted fluent dependency graph corresponding to fluents $movement(P_1, P_2)$ and $interaction(P_1, P_2)$ has level *2* (see Figure 1d). Thus, the FVPs with fluent $movement(P_1, P_2)$, i.e., $movement(P_1,P_2) = gathering$ and $movement(P_1,P_2) = abrupt\_gestures$, and the FVPs with fluent $interaction(P_1,P_2)$, i.e., $interaction(P_1,P_2) = greeting$, $interaction(P_1, P_2) = talking$, have level *2*.

We can devise a local stratification of an event description by following bottom-up the levels of FVPs, as specified in Definition 10. For each level with cyclic dependencies, we introduce an additional stratum per time-point, following an ascending temporal order.

Proposition 2 (Semantics of RTEC$_{fl}$)**.** An event description is a locally stratified logic program. ⧫

According to Proposition 2, RTEC$_{fl}$ supports every event description $\mathcal{E}$ that follows Definition 1. If the dependency graph of $\mathcal{E}$ contains FVPs with the same fluent whose vertices are in different levels of the graph, then these FVPs are assigned the same level, following the definition of FVP level in

Algorithm 1 *compile*($\mathcal{E}$)

1: $G_{\mathcal{E}}^{cdfl} \leftarrow construct\_contracted\_fluent\_dependency\_graph(\mathcal{E})$
2: $level \leftarrow compute\_fvp\_level(G_{\mathcal{E}}^{cdfl})$
3: **for each** rule r **in** $\mathcal{E}$ **do**
4: $\quad F = V \leftarrow get\_fvp\_in\_head(r)$
5: $\quad$ **for each** condition '[not] holdsAt($F' = V'$, T)' **in** the body of r **do**
▷ not is optional.
6: $\quad\quad$ **if** $level[F' = V'] = level[F = V]$ **then**
7: $\quad\quad\quad$ **replace** '[not] holdsAt($F' = V'$, T)'
with '[not] holdsAtCyclic($F' = V'$, T)' **in** r
8: **return** $\mathcal{E}$

RTEC$_{fl}$ (see Definition 10), avoiding the issues described in Section 4.

5.2. Compiler

We developed a compiler that assigns a level to each FVP of an input event description $\mathcal{E}$ and marks the holdsAt body conditions of the rules in $\mathcal{E}$ that must be evaluated with incremental caching, in order to guarantee correct reasoning. The compilation is performed before the commencement of run-time reasoning, in a process transparent to the event description developer. According to Algorithm 1, the compilation steps are the following:

1. First, we derive the levels of the FVPs in the input event description $\mathcal{E}$ by following Definitions 9 and 10, i.e.:
   - We construct the contracted fluent dependency graph $G_{\mathcal{E}}^{cdfl}$ of $\mathcal{E}$ (line 1 of Algorithm 1).
   - Subsequently, we assign a level to each FVP in $\mathcal{E}$ based on the level of the corresponding vertex of $G_{\mathcal{E}}^{cdfl}$ (line 2).
2. The compiler identifies the holdsAt conditions in $\mathcal{E}$ that need to be evaluated with incremental caching. For each holdsAt($F' = V'$, T) or 'not holdsAt($F' = V'$, T)' condition in the body of a rule in $\mathcal{E}$:
   - The compiler checks whether the level of FVP $F' = V'$ is equal to the level of the FVP in the head of the rule (lines 3–6).
   - If this is the case, then the compiler translates holdsAt($F' = V'$, T) (resp. 'not holdsAt($F' = V'$, T)') into holdsAtCyclic($F' = V'$, T) ('not holdsAtCyclic($F' = V'$, T)') (line 7).

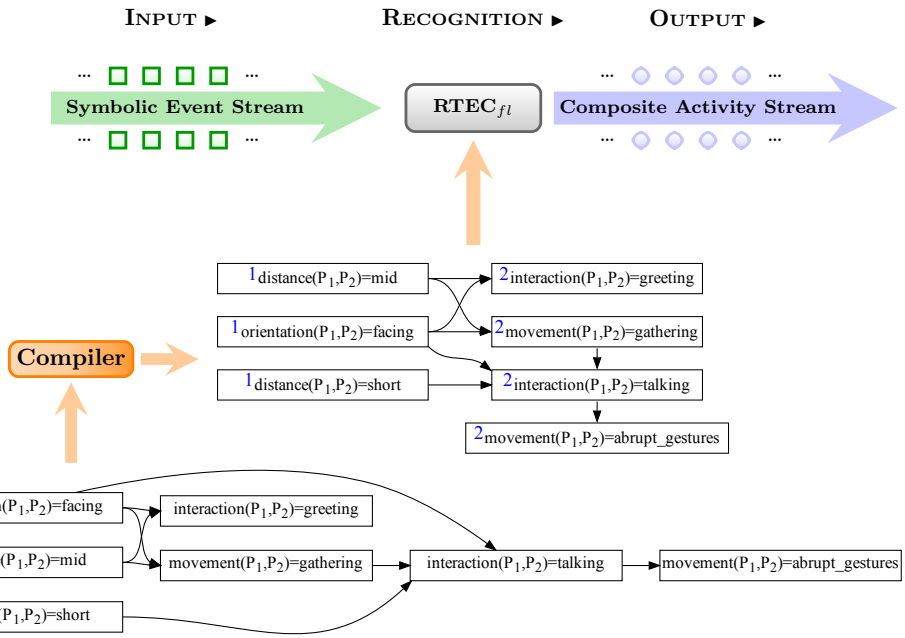


Figure 3: Compilation of event description $\mathcal{E}_3$ into event description $\mathcal{E}_3^*$, and the subsequent use of $\mathcal{E}_3^*$ by RTEC$_{fl}$ for CER. We represent $\mathcal{E}_3$ with the corresponding dependency graph $G_{\mathcal{E}_3}$, and $\mathcal{E}_3^*$ with a version of $G_{\mathcal{E}_3}$ where each vertex is annotated with the level computed by our compiler for the corresponding FVP.

Figure 3 depicts the compilation of event description $\mathcal{E}_3$, resulting in event description $\mathcal{E}_3^*$, and the subsequent use of $\mathcal{E}_3^*$ by RTEC$_{fl}$ for CER. Given $\mathcal{E}_3$ as input, our compiler first constructs the contracted fluent dependency graph of $\mathcal{E}_3$ and then computes the levels of the FVPs in $\mathcal{E}_3$ based on this graph (see lines 1–2 of Algorithm 1). In Figure 3, the level derived for each FVP is displayed in the corresponding vertex of the dependency graph in the output of our compiler. Afterwards, our compiler iterates over the rules in $\mathcal{E}_3$; for each rule, our compiler identifies the holdsAt$(F' = V', T)$ expressions in rules where the level of FVP $F' = V'$ is equal to the level of the FVP in the head of the rule and translates them into holdsAtCyclic$(F' = V', T)$ (see lines 3–7). In other words, when there is an edge $(v_{F_j = V_j}, v_{F_i = V_i})$ in the dependency graph where $F_j = V_j$ and $F_i = V_i$ have the same level, we use holdsAtCyclic for every condition with FVP $F_j = V_j$ that is in the body of a rule with FVP $F_i = V_i$ in its head. In the case of $\mathcal{E}_3$, the edges connecting vertices of FVPs with the same level are $(v_{movement(P_1,P_2)=gathering}, v_{interaction(P_1,P_2)=talking})$ and

$(v_{interaction(P_1,P_2)=talking}, v_{movement(P_1,P_2)=abrupt\_gestures})$ (see the dependency graph in the output of our compiler in Figure 3). As a result, the holdsAt$(movement(P_1, P_2) = gathering, T)$ expressions in rules with FVP $interaction(P_1, P_2) = talking$ in their head (i.e., rules (9)–(10)), and the holdsAt$(interaction(P_1, P_2) = talking, T)$ expressions in rules with FVP $movement(P_1, P_2) = abrupt\_gestures$ in their head (i.e., rules (12)–(13)), are translated into holdsAtCyclic. These translations lead to event description $\mathcal{E}^*_\mathcal{I}$, which may subsequently be supplied to RTEC$_{fl}$ in order to perform CER. At run-time, RTEC$_{fl}$ evaluates the conditions with holdsAtCyclic using incremental caching (recall the second case presented in Section 3.2) and the conditions with holdsAt using the interval retrieval operation (see the first case of Section 3.2). A further discussion on run-time reasoning is presented in Section 8.

6. Formal Properties

6.1. Correctness

We formulate the requirements for the correctness of our compiler with respect to its task, i.e., the identification of the holdsAt conditions in the rules of an event description that may require cyclic processing, and should thus be translated into holdsAtCyclic. Suppose that we translated every holdsAt condition into holdsAtCyclic, even though only a subset of these conditions may require cyclic processing. Such a translation should not be considered correct, as, since processing holdsAtCyclic is more costly than processing holdsAt, it may lead to redundant computations that could have been avoided by translating only the holdsAt conditions that require cyclic processing. On the other end of the spectrum, translating none of the holdsAt conditions into holdsAtCyclic may also be incorrect, because it leads to erroneous computations for the holdsAt conditions that require cyclic processing. Therefore, our compiler is correct if it translates a holdsAt condition into holdsAtCyclic iff this condition requires cyclic processing.

The proposed compiler is sound, meaning that every holdsAt condition in a rule of an event description that is compiled into holdsAtCyclic requires indeed cyclic processing. In other words, soundness implies that no holdsAt condition that does not require cyclic processing is identified by our compiler as requiring such processing, i.e., our compiler does not idenify a 'false positive'. Moreover, our compiler is complete in the sense that every holdsAt condition in a rule of an event description that requires cyclic processing is

compiled into holdsAtCyclic. In other words, completeness implies that there is no holdsAt condition that requires cyclic processing, but is not identified by our compiler as requiring such processing, which would constitute a 'false negative'. In the following, we prove that our compiler is correct, i.e., sound and complete. We start with two corollaries of our definition for FVP level in RTEC$_{fl}$ (Definition 10). First, according to Definition 10, two FVPs with the same fluent, such as $F = V$ and $F = V'$, where $V \neq V'$, have the same level as the vertex of the contracted fluent dependency graph that corresponds to the SCC of the fluent dependency graph that includes vertex v_F. Thus, $F = V$ and $F = V'$ have the same level.

Corollary 1. Based on Definition 10, FVPs with the same fluent have the same level. △

Second, consider a rule r that has FVP $F = V$ in its head and condition holdsAt$(F' = V', T)$ in its body. Based on Definition 3, there is edge $(v_{F'=V'}, v_{F=V})$ in the dependency graph. Considering the corresponding fluent dependency graph, there are two options, i.e., vertices $v_{F'}$ and v_F are either in the same SCC or in different SCCs of the fluent dependency graph. In the former case, according to Definition 10, $F = V$ and $F' = V'$ have the same level. In the latter case, suppose that $v_{F'}$ and v_F are in SCCs S' and S, respectively. Since we have edge $(v_{F'=V'}, v_{F=V})$ in the dependency graph, there is edge $(v_{S'}, v_S)$ in the contracted fluent dependency graph, which is implies that $F = V$ has a higher level than $F' = V'$ (Definition 10). Therefore, the level of $F = V$ is greater or equal to the level of $F' = V'$.

Corollary 2. Consider a rule r that has FVP $F = V$ in its head and a holdsAt$(F' = V', T)$ condition in its body. Based on Definition 10, the level of $F = V$ is greater or equal to the level of $F' = V'$. △

We start by proving that our compiler is sound, and then proceed to a proof of completeness.

Proposition 3 (Soundness)**.** Every holdsAt condition in a rule of an event description that is compiled into holdsAtCyclic requires cyclic processing. ⧫

Proof. Consider an event description with a rule r, having FVP $F = V$ in its head and a holdsAt$(F' = V', T)$ condition in its body, where F and F' are different fluents. Suppose that, given this event description as input, Algorithm 1 replaces condition holdsAt$(F' = V', T)$ of rule r with condition holdsAtCyclic$(F' = V', T)$. It suffices to prove that it is possible for an execution of RTEC$_{fl}$ to reach this condition of rule r at a time when the intervals

of $F' = V'$ are not present in the cache.

The dependency of $F = V$ on $F' = V'$ that is described in rule r leads to edge $(v_{F'=V'}, v_{F=V})$ in the dependency graph (Definition 3) and edge $(v_{F'}, v_F)$ in the fluent dependency graph (Definition 8). Moreover, since condition $\mathsf{holdsAt}(F' = V', T)$ of rule r has been replaced with condition $\mathsf{holdsAtCyclic}(F' = V', T)$, it holds that FVPs $F=V$ and $F' = V'$ have the same level (see lines 6–7 of Algorithm 1). Therefore, based on Definition 10, the vertices v_F and $v_{F'}$ of the fluent dependency graph of the event description are in the same SCC, which implies that there is a path from vertex v_F to vertex $v_{F'}$ in the fluent dependency graph. As a result, there are some values V_0 and V_0' for fluents F and F' such that there is a path from vertex $v_{F=V_0}$ to vertex $v_{F'=V_0'}$ in the dependency graph (see Definition 8). Based on Corollary 1, FVPs $F = V_0$ and $F' = V_0'$ have the same level as FVPs $F = V$ and $F' = V'$. Thus, due to the bottom-up processing order we follow, RTEC$_{fl}$ may start processing any one of the FVPs $F' = V'$, $F = V$, $F' = V_0'$ and $F = V_0$, while the remaining ones have not been processed yet.

Suppose that RTEC$_{fl}$ starts by processing $F' = V'$, as an attempt to compute and cache the intervals of $F' = V'$ before processing rule r. In order to compute the terminations of $F' = V'$, we have to compute the initiations of $F' = V_0'$, which, based on the path from vertex $v_{F=V_0}$ to vertex $v_{F'=V_0'}$ in the dependency graph, depend on $F = V_0$. Subsequently, in order to evaluate whether $F = V_0$ holds, we need to compute the initiations and the terminations of $F = V_0$. These terminations of $F = V_0$ include the initiations of $F = V$. In order to process $F=V$, we need to evaluate rule r, whose body includes condition $\mathsf{holdsAtCyclic}(F' = V', T)$. In the aforementioned scenario, $\mathsf{holdsAtCyclic}(F' = V', T)$ is invoked at a time when the intervals of $F' = V'$ are not present in the cache, proving the proposition. □

Proposition 4 (Completeness)**.** Every holdsAt condition in a rule of an event description that requires cyclic processing is compiled into holdsAtCyclic. ♦

Proof. Consider an event description containing a rule r that has FVP $F = V$ in its head and a $\mathsf{holdsAt}(F' = V', T)$ condition in its body. Suppose that, given this event description as input, Algorithm 1 does not replace condition $\mathsf{holdsAt}(F' = V', T)$ of rule r with condition $\mathsf{holdsAtCyclic}(F' = V', T)$. It suffices to prove that, when an execution of RTEC$_{fl}$ reaches this condition of rule r, the intervals of $F' = V'$ are present in the cache.

Based on rule r, there is an edge pointing from $v_{F'=V'}$ to $v_{F=V}$ in the dependency graph (Definition 3), which implies that there is a vertex

pointing from $v_{F'}$ to v_F in the fluent dependency graph (Definition 8). We first show that there is no path from v_F to $v_{F'}$ in the fluent dependency graph. Since we have not replaced condition $\mathsf{holdsAt}(F'=V', T)$ of rule r with $\mathsf{holdsAtCyclic}(F'=V', T)$, based on lines 6–7 of Algorithm 1, it holds that FVPs $F=V$ and $F'=V'$ do not have the same level. Thus, based on Corollary 2, $F=V$ has a higher level than $F'=V'$. According to the definition of FVP level in RTEC$_{fl}$ (Definition 10), this implies that the vertices v_F and $v_{F'}$ are in different SCCs of the fluent dependency graph. Therefore, since the fluent dependency graph contains edge $(v_{F'}, v_F)$, there is no path from vertex v_F to vertex $v_{F'}$ in the fluent dependency graph, as, in that case, v_F and $v_{F'}$ would have been in the same SCC. Based on Definition 8, this implies that there is no FVP with fluent F that depends on an FVP with fluent F'. We will use this result to prove that condition $\mathsf{holdsAt}(F'=V', T)$ of r is always invoked at a time when the intervals of $F'=V'$ are present in the cache.

Since $F'=V'$ has a lower level than $F=V$, RTEC$_{fl}$ processes $F'=V'$ before $F=V$, by computing the initiations and the terminations of $F'=V'$. Since there is no FVP with fluent F' that depends on an FVP with fluent F, the initiatedAt and the terminatedAt rules of $F'=V'$, as well as the initiatedAt rules of any other FVP with fluent F', which imply the termination of $F'=V'$, do not contain any condition that leads to the computation of rule r. Therefore, it is possible to compute the initiations and the terminations of $F'=V'$ without rule r. This is always achieved by RTEC$_{fl}$ because $F'=V'$ is assigned a lower level than $F=V$, and is thus processed before $F=V$. As a result, $\mathsf{holdsAt}(F'=V', T)$ is always invoked at a time when $F'=V'$ has been processed and its intervals are present in the cache, proving the proposition. □

Propositions 3 and 4 demonstrate that our compiler is sound and complete. Thus, our compiler is correct, i.e., it identifies, and translates into holdsAtCyclic, the holdsAt conditions of an event description that may require cyclic processing at run-time, and no other conditions.

6.2. Optimality

An event description generated by our compiler is *holdsAt-optimal* in the following sense.

Definition 11 (holdsAt-optimal Event Description)**.** We say that an event description is holdsAt-optimal if replacing any holdsAtCyclic condition of the

event description with holdsAt leads to incorrect reasoning. ■

For the event descriptions generated by our compiler, holdsAt-optimality follows directly from the soundness of the compiler (Proposition 3). According to this proposition, our compiler translates holdsAt$(F' = V', T)$ into holdsAtCyclic$(F' = V', T)$ only if this condition may be invoked at a time when the intervals of $F' = V'$ are not cached. Therefore, using holdsAt$(F' = V', T)$ instead of holdsAtCyclic$(F' = V', T)$ may lead to a situation where condition holdsAt$(F' = V', T)$ is invoked before having computed and cached the intervals of $F' = V'$, which results in erroneous reasoning.

Corollary 3 (holdsAt-optimality of Compiled Event Descriptions)**.** Given an input event description without holdsAtCyclic conditions, Algorithm 1 returns a holdsAt-optimal event description. ⧫

Note that our compiler operates under the assumption that the input event description does not contain holdsAtCyclic, i.e., the event description developer is not expected to use this predicate.

We are interested in holdsAt-optimality because it implies that the event description is minimal with respect to the use of holdsAtCyclic conditions. In other words, holdsAt conditions, which are computed more efficiently than their corresponding holdsAtCyclic counterparts [48], are used in the event description as much as possible, while guaranteeing reasoning correctness.

6.3. Complexity

Proposition 5 (Complexity of Compiler)**.** The worst-case time complexity of compiling an event description with n_f fluents and n_c holdsAt body conditions is $O(n_f+n_c)$. ⧫

Proof. The first step of Algorithm 1 is to construct the contracted fluent dependency graph of the input event description (see line 1). We start by building its dependency graph. To do this, we need to inspect every holdsAt condition in the event description, as each one corresponds to an edge of the dependency graph (see Definition 3). This process requires n_c steps, where n_c is the number of holdsAt conditions in the event description, and thus has a cost of $O(n_c)$. Subsequently, we construct the fluent dependency graph of the event description, which involves one iteration over the edges of the dependency graph, as each one corresponds to an edge in the fluent dependency graph (see Definition 8). In the worst case, we have one edge in the dependency graph for each holdsAt condition in the event description,

and thus the cost of this step is $O(n_c)$. Afterwards, we contract the SCCs of the fluent dependency graph, leading to the contracted fluent dependency graph. We identify these SCCs using Tarjan's algorithm [61] on the fluent dependency graph. The complexity of this process is linear to the number of vertices and to the number of edges in the fluent dependency graph, which are bounded, respectively, by the number of fluents n_f and the number of holdsAt conditions n_c in the event description (Definition 8). Thus, the cost of this step is $O(n_f+n_c)$. Therefore, the overall cost of constructing the contracted fluent dependency graph of an event description is $O(n_f+3n_c)$.

Subsequently, Algorithm 1 computes the levels of the FVPs in the event description based on its contracted fluent dependency graph (line 2). To do this, we compute a topological sort of the vertices in the contracted fluent dependency graph using Kahn's algorithm [37]. We employ a modified version of this algorithm, where, at the time of adding a vertex v to the topological ordering, we derive its level as the maximum level of the vertices with outgoing edges pointing to v, plus *1*. The levels of these vertices have been defined earlier in the procedure, since a topological sort orders the vertices of an acyclic graph bottom-up. The cost of this algorithm is linear to the number of vertices and to the number of edges in the contracted fluent dependency graph, i.e., $O(n_f+n_c)$. Finally, in lines 3–7, Algorithm 1 iterates over each holdsAt condition in each rule of the event description. Thus, the cost of these operations is $O(n_c)$.

Overall, the cost of our compiler is $O(2n_f+5n_c)$, which, after simplifications, leads to a worst-case time complexity of $O(n_f+n_c)$. □

We tested the compiler of RTEC$_{fl}$ on event descriptions from various CER applications, including human activity recognition [6], city transport management [7] and maritime situational awareness [53, 54]. Moreover, we have used our compiler in applications that involve the monitoring of the normative positions of agents in multi-agent systems, such as e-commerce [59] and voting protocols [55]. In all cases, the compilation time amounted to a few milliseconds, and thus we do not show these times here. The compiler is available with the code of RTEC$_{fl}$[1].

7. Experimental Evaluation

7.1. Experimental Setup

We evaluated RTEC$_{fl}$ on event descriptions that are not supported by RTEC$_{\circ}$. We employed an event description for human activity recognition,

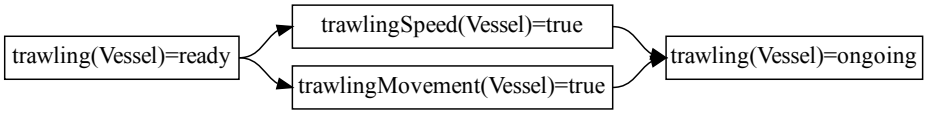


Figure 4: Fragment of the dependency graph of our event description for maritime situational awareness.

including rules (2)–(15), and an event description for maritime situational awareness, where FVPs are used to specify various types of dangerous, suspicious or illegal vessel activities, such as fishing in an environmentally protected area [54]. Our maritime event description cannot be processed by RTEC$_\circ$ because it includes the following rules:

$$\begin{aligned}&\mathsf{initiatedAt}(\mathit{trawling}(\mathit{Vessel}) = \mathit{ready},\ T) \leftarrow \\ &\quad \mathsf{happensAt}(\mathit{entersArea}(\mathit{Vessel}, \mathit{fishingArea}),\ T), \\ &\quad \mathit{vesselType}(\mathit{Vessel}, \mathit{fishingVessel}).\end{aligned} \tag{16}$$

$$\begin{aligned}&\mathsf{initiatedAt}(\mathit{trawlingSpeed}(\mathit{Vessel}) = \mathit{true},\ T) \leftarrow \\ &\quad \mathsf{happensAt}(\mathit{velocity}(\mathit{Vessel}, \mathit{Speed}),\ T), \\ &\quad \mathit{isTrawlingSpeed}(\mathit{Speed}), \\ &\quad \mathsf{holdsAt}(\mathit{trawling}(\mathit{Vessel}) = \mathit{ready},\ T).\end{aligned} \tag{17}$$

$$\begin{aligned}&\mathsf{initiatedAt}(\mathit{trawlingMovement}(\mathit{Vessel}) = \mathit{true},\ T) \leftarrow \\ &\quad \mathsf{happensAt}(\mathit{changeInHeading}(\mathit{Vessel}),\ T), \\ &\quad \mathsf{holdsAt}(\mathit{trawling}(\mathit{Vessel}) = \mathit{ready},\ T).\end{aligned} \tag{18}$$

$$\begin{aligned}&\mathsf{initiatedAt}(\mathit{trawling}(\mathit{Vessel}) = \mathit{ongoing},\ T) \leftarrow \\ &\quad \mathsf{happensAt}(\mathit{movementChange}(\mathit{Vessel}),\ T), \\ &\quad \mathsf{holdsAt}(\mathit{trawlingSpeed}(\mathit{Vessel}),\ T), \\ &\quad \mathsf{holdsAt}(\mathit{trawlingMovement}(\mathit{Vessel}),\ T).\end{aligned} \tag{19}$$

Rules (16)–(19) constitute a partial specification for the fishing activity 'trawling'. Rule (16) states that a vessel is ready to perform this fishing activity if it is a fishing vessel and it has entered a fishing area. Rule (17) determines when a vessel sails at a speed that is expected for trawling, while rule (18) is used to denote the time periods during which the vessel is performing consecutive turns, as is typical when trawling. Both these rules require that the vessel is ready for trawling, i.e., $\mathit{trawling}(\mathit{Vessel}) = \mathit{ready}$ holds. Rule (19) states that a trawling activity is ongoing when the vessel is sailing at a speed and with a direction that is typical for trawling.

Figure 4 depicts a fragment of the dependency graph of our mairitime

event description that captures the dependencies in rules (16)–(19). Following these dependencies and the definition of FVP level in RTEC$_\circ$ (Definition 7), FVPs $trawling(Vessel) = ready$ and $trawling(Vessel) = ongoing$ have different levels. This is because $trawling(Vessel) = ongoing$ depends on FVPs $trawlingSpeed(Vessel) = \mathsf{true}$ and $trawlingMovement(Vessel) = \mathsf{true}$ which in turn depend on $trawling(Vessel) = ready$. As a result, based on Definition 7, rules (16)–(19) specify two FVPs with the same fluent and different levels, and thus are not supported in RTEC$_\circ$. In contrast, RTEC$_{fl}$ supports rules (16)–(19); following Definition 10, FVPs $trawling(Vessel) = ready$, $trawlingSpeed(Vessel) = \mathsf{true}$, $trawlingMovement(Vessel) = \mathsf{true}$ and $trawling(Vessel) = ongoing$ are assigned the same level, and are thus handled by the cycle processing module of RTEC$_{fl}$.

For human activity recognition, we used synthetic datasets produced by a custom generator via simulating scenarios where multiple people move around in a bounded area, and may engage in meeting activities. For maritime situational awareness, we employed an event stream that was derived by processing Automatic Identification System (AIS) signals, containing information about vessels' location, speed and heading. We used a publicly available dataset[2] including 18M AIS signals, emitted by 5K vessels sailing around the port of Brest, France, between October 2015–March 2016.

We compared RTEC$_{fl}$ with jREC$^{\text{rbt}}$, i.e., an implementation of the 'Reactive Event Calculus' [49] that employs indexing for manipulating lists of events and FVP intervals efficiently [29], and Fusemate, a stream reasoning framework that integrates the Event Calculus with a description logic in logic programming [11]. We ran RTEC$_{fl}$ using SWI-9.2 Prolog, jREC$^{\text{rbt}}$ using Java OpenJDK 18, and Fusemate using Scala 2.13, with a time limit of 30 minutes. We used a single core of a desktop PC running Ubuntu 22, with Intel Core i7-4770 CPU @3.40GHz and 16GB RAM.

7.2. Experimental Results

Our first set of experiments compared RTEC$_{fl}$, jREC$^{\text{rbt}}$ and Fusemate on the task of detecting the composite activities in human activity recognition. We used human activity recognition datasets whose size ranged from 400 to 3200 events. Figure 5 (left) presents the reasoning times of RTEC$_{fl}$, jREC$^{\text{rbt}}$ and Fusemate on these datasets. Our results show that RTEC$_{fl}$ was able to

[2] `https://zenodo.org/records/1167595`

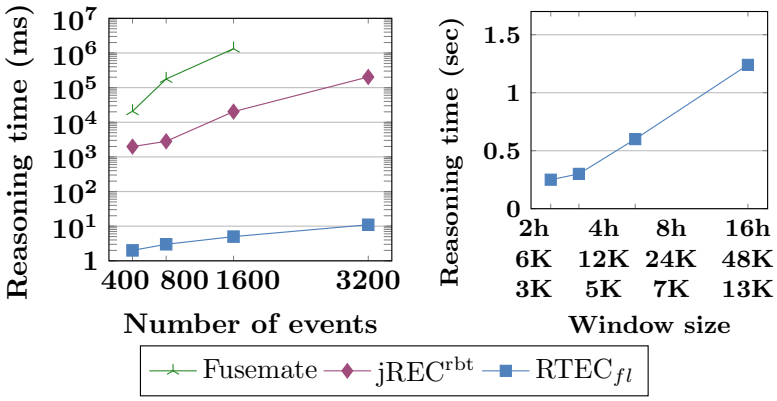


Figure 5: Stream reasoning in human activity recognition (left), and in maritime situational awareness (right). The horizontal axis of the right diagram shows the size of stream in terms of hours (top), number of input events (middle) and number of computed FVP intervals (bottom).

compute the maximal intervals of FVPs corresponding to composite human activities, such as $movement(P_1, P_2) = abrupt\_gestures$, in a few milliseconds, while jREC$^{\text{rbt}}$ and Fusemate required much more time to produce the same results. In our most demanding case where 3200 input events needed to be processed, RTEC$_{fl}$ required about 10 milliseconds to compute all composite activities in the dataset, while jREC$^{\text{rbt}}$ needed about 3 minutes to complete the same task, and we terminated the execution of Fusemate early, as it lasted for more than 30 minutes. Contrary to RTEC$_{fl}$, jREC$^{\text{rbt}}$ and Fusemate do not use incremental caching to process FVP hierarchies with cyclic dependencies, leading to much less efficient reasoning compared to RTEC$_{fl}$.

In our second set of experiments, we instructed RTEC$_{fl}$ to detect instances of composite maritime activities in our maritime dataset. We did not include jREC$^{\text{rbt}}$ and Fusemate in these experiments because they do scale on our maritime dataset due to its volume. We ran RTEC$_{fl}$ using windows, i.e., finite chunks of the input event stream that are being processed sequentially by RTEC$_{fl}$, of increasing size, spanning from 2 hours to 16 hours. Figure 5 (right) depicts the average reasoning times of RTEC$_{fl}$ per window. For the largest window size spanning 16 hours, including, on average, 48 thousand input events, RTEC$_{fl}$ was able to compute an average of 13 thousand maximal intervals for FVPs corresponding to composite maritime activities. These results demonstrate that it may be possible to use RTEC$_{fl}$ at scale, on event

streams stemming from demanding real-world applications.

8. Discussion

We proposed RTEC$_{fl}$, an extension of RTEC$_\circ$, which detects composite activities based on their Event Calculus definitions, in order to support arbitrary such definitions. We described the syntax and the semantics of RTEC$_{fl}$, demonstrating that activity specifications in RTEC$_{fl}$ are locally stratified logic programs. Afterwards, we proposed a compiler for RTEC$_{fl}$, identifying the conditions of activity definitions that may be evaluated with an efficient cache operation, without sacrificing correctness, with the goal of improving reasoning efficiency at run-time. We proved the correctness of our compiler and outlined its time complexity. Moreover, we demonstrated that a set of activity definitions generated by our compiler is optimal, i.e., RTEC$_{fl}$ does not perform any redundant computations when reasoning with such a set of definitions. We evaluated RTEC$_{fl}$ empirically on synthetic and real event streams from human activity recognition and maritime situational awareness, and compared it with two state-of-the-art Event Calculus-based frameworks, demonstrating the high reasoning efficiency and scalability of RTEC$_{fl}$.

RTEC$_{fl}$ follows RTEC$_\circ$ and processes FVPs in ascending FVP level order. When processing a rule that includes a $\mathsf{holdsAtCyclic}(F' = V', T)$ condition, RTEC$_{fl}$ computes the changes in the value of F' between T_{leq} and T, where T_{leq} is the last time-point before T where the truth value of $\mathsf{holdsAt}(F' = V', T_{leq})$ has been evaluated and cached. In the worst-case, the cost of this process is $\mathcal{O}(\omega k)$, where ω is the size of the window and k is the cost of computing whether an FVP is initiated or terminated at a given time-point (see [7] for an estimation of k). This is the same incremental caching technique as the one used in RTEC$_\circ$, thus yielding the same cost [48]. In the case of a $\mathsf{holdsAt}(F' = V', T)$ condition, RTEC$_{fl}$ retrieves the maximal intervals of $F' = V'$ from its cache and checks whether T belongs to one of the retrieved intervals. Since the cached intervals are temporally sorted, this is achieved with a binary search, while the number of cached intervals of $F' = V'$ is bounded by ω. Therefore, the cost of an interval retrieval operation in RTEC$_{fl}$ is $\mathcal{O}(log(\omega))$, which is the same as the cost of this operation in RTEC$_\circ$. As a result, RTEC$_{fl}$ yields the same worst-case time complexity as RTEC$_\circ$, while supporting a wider range of temporal specifications. By following Definition 10 for FVP level, RTEC$_{fl}$ reasons with incremental caching

only when it is necessary, i.e., only when the required intervals may not be present in the cache. The code of RTEC$_{fl}$ is publicly available[1].

In the future, we aim to employ Large Language Models (LLM)s for compiling natural language to RTEC$_{fl}$ event descriptions. The reasoning abilities of LLMs, particularly in multi-step logical reasoning tasks, remain constrained [34]. Such reasoning tasks become even more demanding when we need to reason over sequences of events that evolve over time, like, e.g., streams of vessel position signals. However, LLMs are increasingly able to generate formal language expressions, such as SQL queries [31], Answer Set Programming (ASP) specifications [35], and First-Order Logic formula [18], based on natural language descriptions of these expressions. In [41, 40] we presented an initial version of a prompting technique with which a domain expert, such as an expert from the maritime domain, may generate RTEC activity definitions with the use of LLMs. In the future, we will refine this technique in order to minimise further the effort of domain experts, as well as improve the responses of LLMs in an automated way with the use of the compiler of RTEC.

Acknowledgements

This work was supported by the EU-funded CREXDATA project (No 101092749).

References

[1] Aghasadeghi, A., den Bussche, J.V., Stoyanovich, J., 2024. Temporal graph patterns by timed automata. VLDB J. 33, 25–47.

[2] Alevizos, E., Skarlatidis, A., Artikis, A., Paliouras, G., 2017. Probabilistic complex event recognition: A survey. Commun. ACM 50, 71:1–71:31.

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

[4] Arasu, A., Babu, S., Widom, J., 2006. The CQL continuous query language: semantic foundations and query execution. VLDB J. 15, 121–142.

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

[6] Artikis, A., Sergot, M.J., Paliouras, G., 2010. A logic programming approach to activity recognition, in: EIMM Workshop in MM, pp. 3–8.

[7] Artikis, A., Sergot, M.J., Paliouras, G., 2015. An event calculus for event recognition. IEEE Trans. Knowl. Data Eng. 27, 895–908.

[8] Artikis, A., Zissis, D. (Eds.), 2021. Guide to Maritime Informatics. Springer.

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

[10] Bai, Y., Thakkar, H., Wang, H., Luo, C., Zaniolo, C., 2006. A data stream language and system designed for power and extensibility, in: CIKM, pp. 337–346.

[11] Baumgartner, P., 2021. Combining event calculus and description logic reasoning via logic programming, in: FroCoS, pp. 98–117.

[12] Bazoobandi, H.R., Beck, H., Urbani, J., 2017. Expressive stream reasoning with laser, in: ISWC, pp. 87–103.

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

[14] Beck, H., Eiter, T., Folie, C., 2017. Ticker: A system for incremental asp-based stream reasoning. Theory Pract. Log. Program. 17, 744–763.

[15] Bragaglia, S., Chesani, F., Mello, P., Montali, M., Torroni, P., 2012. Reactive event calculus for monitoring global computing applications, in: Logic Programs, Norms and Action - Essays in Honor of Marek J. Sergot on the Occasion of His 60th Birthday, pp. 123–146.

[16] Brandt, S., Kalayci, E.G., Ryzhikov, V., Xiao, G., Zakharyaschev, M., 2018. Querying log data with metric temporal logic. J. Artif. Intell. Res. 62, 829–877.

[17] Bromuri, S., Urovi, V., Stathis, K., 2010. icampus: A connected campus in the ambient event calculus. Int. J. Ambient Comput. Intell. 2, 59–65.

[18] Brunello, A., Ferrarese, R., Geatti, L., Marzano, E., Montanari, A., Saccomanno, N., 2024. Evaluating llms capabilities at natural language to logic translation: A preliminary investigation, in: Porello, D., Vinci, C., Zavatteri, M. (Eds.), Short Paper Proceedings of the 6th International Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, OVERLAY 2024, Bolzano, Italy, November 28-29, 2024, CEUR-WS.org. pp. 103–110. URL: `https://ceur-ws.org/Vol-3904/paper13.pdf`.

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

[20] Calimeri, F., Manna, M., Mastria, E., Morelli, M.C., Perri, S., Zangari, J., 2021. I-dlv-sr: A stream reasoning system based on I-DLV. Theory Pract. Log. Program. 21, 610–628.

[21] Cervesato, I., Montanari, A., 2000. A calculus of macro-events: Progress report, in: TIME, pp. 47–58.

[22] Chaudet, H., 2006. Extending the event calculus for tracking epidemic spread. Artif. Intell. Medicine 38, 137–156.

[23] Chittaro, L., Montanari, A., 1996. Efficient temporal reasoning in the cached event calculus. Comput. Intell. 12, 359–382.

[24] Clark, K.L., 1977. Negation as failure, in: Logic and Data Bases, Plemum Press. pp. 293–322.

[25] Cugola, G., Margara, A., 2010. TESLA: A formally defined event specification language, in: DEBS, ACM. pp. 50–61.

[26] Cugola, G., Margara, A., 2012. Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. 44.

[27] Dousson, C., Maigat, P.L., 2007. Chronicle recognition improvement using temporal focusing and hierarchisation, in: IJCAI, pp. 324–329.

[28] Eiter, T., Ogris, P., Schekotihin, K., 2019. A distributed approach to LARS stream reasoning (system paper). Theory Pract. Log. Program. 19, 974–989.

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

[30] Galton, A., Augusto, J.C., 2002. Two approaches to event definition, in: DEXA, pp. 547–556.

[31] Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., Zhou, J., 2024. Text-to-sql empowered by large language models: A benchmark evaluation. Proc. VLDB Endow. 17, 1132–1145.

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

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

[34] Ishay, A., Lee, J., 2025. LLM+AL: bridging large language models and action languages for complex reasoning about actions, in: Walsh, T., Shah, J., Kolter, Z. (Eds.), AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA, AAAI Press. pp. 24212–24220. URL: https://doi.org/10.1609/aaai.v39i23.34597, doi:10.1609/AAAI.V39I23.34597.

[35] Ishay, A., Yang, Z., Lee, J., 2023. Leveraging large language models to generate answer set programs, in: KR, pp. 374–383.

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

[37] Kahn, A.B., 1962. Topological sorting of large networks. Commun. ACM 5, 558–562.

[38] Katzouris, N., Paliouras, G., Artikis, A., 2023. Online learning probabilistic event calculus theories in answer set programming. Theory Pract. Log. Program. 23, 362–386.

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

[40] Kouvaras, A., Mantenoglou, P., Artikis, A., . Prompting llms for the run-time event calculus, in: 32nd International Symposium on Temporal Representation and Reasoning, TIME 2025.

[41] Kouvaras, A., Mantenoglou, P., Artikis, A., 2025. Generating activity definitions with large language models, in: Simitsis, A., Kemme, B., Queralt, A., Romero, O., Jovanovic, P. (Eds.), Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025, OpenProceedings.org. pp. 1005–1013. URL: https://doi.org/10.48786/edbt.2025.83, doi:10.48786/EDBT.2025.83.

[42] Kowalski, R.A., Sergot, M.J., 1986. A logic-based calculus of events. New Gener. Comput. 4, 67–95.

[43] Laptev, N., Mozafari, B., Mousavi, H., Thakkar, H., Wang, H., Zeng, K., Zaniolo, C., 2016. Extending relational query languages for data streams, in: Data Stream Management. Springer, pp. 361–386.

[44] Lee, J., Palla, R., 2012. Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. J. Artif. Intell. Res. 43, 571–620.

[45] Li, M., Mani, M., Rundensteiner, E.A., Lin, T., 2011. Complex event pattern detection over streams with interval-based temporal semantics, in: DEBS, ACM. pp. 291–302.

[46] Mantenoglou, P., Artikis, A., 2024. Extending the range of temporal specifications of the run-time event calculus, in: Sala, P., Sioutis, M., Wang, F. (Eds.), 31st International Symposium on Temporal Representation and Reasoning, TIME 2024, October 28-30, 2024, Montpellier, France, Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

pp. 6:1–6:14. URL: https://doi.org/10.4230/LIPIcs.TIME.2024.6, doi:10.4230/LIPICS.TIME.2024.6.

[47] Mantenoglou, P., Kelesis, D., Artikis, A., 2023. Complex event recognition with allen relations, in: KR, pp. 502–511.

[48] Mantenoglou, P., Pitsikalis, M., Artikis, A., 2022. Stream reasoning with cycles, in: KR, pp. 544–553.

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

[50] Paschke, A., 2005. ECA-RuleML: An Approach combining ECA Rules with Temporal Interval-Based KR Event/Action Logics and Transactional Update Logics. Technical Report 11. TU München.

[51] Paschke, A., 2006. Eca-ruleml: An approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics. CoRR abs/cs/0610167.

[52] Paschke, A., Bichler, M., 2008. Knowledge representation concepts for automated SLA management. Decis. Support Syst. 46, 187–205.

[53] Patroumpas, K., Artikis, A., Katzouris, N., Vodas, M., Theodoridis, Y., Pelekis, N., 2015. Event recognition for maritime surveillance, in: EDBT, pp. 629–640.

[54] Pitsikalis, M., Artikis, A., Dreo, R., Ray, C., Camossi, E., Jousselme, A., 2019. Composite event recognition for maritime monitoring, in: DEBS, pp. 163–174.

[55] Pitt, J., Kamara, L., Sergot, M., Artikis, A., 2006. Voting in multi-agent systems. Comput. J. 49, 156–170.

[56] Poppe, O., Lei, C., Rundensteiner, E.A., Maier, D., 2019. Event trend aggregation under rich event matching semantics, in: SIGMOD, pp. 555–572.

[57] Przymusinski, T.C., 1988. On the declarative semantics of deductive databases and logic programs, in: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, pp. 193–216.

[58] Shahid, N.S., O'Keeffe, D., Stathis, K., 2023. A knowledge representation framework for evolutionary simulations with cognitive agents, in: ICTAI, IEEE. pp. 361–368.

[59] Sirbu, M., 1997. Credits and debits on the Internet. IEEE Spectrum 34, 23–29.

[60] Srinivasan, A., Bain, M., Baskar, A., 2022. Learning explanations for biological feedback with delays using an event calculus. Mach. Learn. 111, 2435–2487.

[61] Tarjan, R.E., 1972. Depth-first search and linear graph algorithms. SIAM J. Comput. 1, 146–160.

[62] Terry, D.B., Goldberg, D., Nichols, D.A., Oki, B.M., 1992. Continuous queries over append-only databases, in: SIGMOD, pp. 321–330.

[63] Tsilionis, E., Artikis, A., Paliouras, G., 2022. Incremental event calculus for run-time reasoning. J. Artif. Intell. Res. 73, 967–1023.

[64] Walega, P.A., Kaminski, M., Grau, B.C., 2019. Reasoning over streaming data in metric temporal datalog, in: AAAI, pp. 3092–3099.

[65] Walega, P.A., Kaminski, M., Wang, D., Grau, B.C., 2023. Stream reasoning with datalogmtl. J. Web Semant. 76, 100776.

[66] White, W.M., Riedewald, M., Gehrke, J., Demers, A.J., 2007. What is "next" in event processing?, in: PODS, pp. 263–272.

[67] Zaniolo, C., 2012. Logical foundations of continuous query languages for data streams, in: Datalog in Academia and Industry.

[68] Zervoudakis, P., Kondylakis, H., Spyratos, N., Plexousakis, D., 2021. Query rewriting for incremental continuous query evaluation in HIFUN. Algorithms 14, 149.

[69] Zhao, B., van der Aa, H., Nguyen, T.T., Nguyen, Q.V.H., Weidlich, M., 2021. EIRES: efficient integration of remote data in event stream processing, in: SIGMOD, pp. 2128–2141.