Temporal Specification Optimisation for the Event Calculus

Periklis Mantenoglou[1], **Alexander Artikis**[1,2]

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

Abstract

Temporal pattern matching tasks require the detection of situations of interest based on streams of symbolic events. The Run-Time Event Calculus (RTEC) is a formal framework that represents situations of interest as time-varying properties called 'fluents'. Temporal patterns often express 'Boolean combinations' of situations; RTEC features two types of fluents that may model such patterns: 'simple' and 'statically determined'. A simple fluent representation, however, is exponentially larger and more expensive to reason with than the corresponding statically determined fluent one. We formally identify the class of simple fluent definitions that can be translated into statically determined fluent definitions. Moreover, we present a compiler for the translation, and a reproducible empirical evaluation on real applications.

Introduction

Temporal pattern matching involves the detection of situations of interest based on streams of symbolic events. In composite event recognition, e.g., we need to reason over streams of time-stamped, symbolic events in order to detect instances of composite activities (Giatrakos et al. 2020). Temporal pattern matching requires a formal language that defines a set of operators between events and determines the syntactical form of the patterns defining situations of interest. These patterns often express 'Boolean combinations' of situations, featuring conjunction, disjunction and negation operators. In human activity recognition, e.g., we may say that two people are meeting when they are close to each other *and* one of them is 'active', i.e., moving her arms while in the same position. Another way of defining a situation of interest is through the law of inertia, i.e., the situation is initiated based on pattern satisfaction, and then persists through time, as long as a pattern terminating the situation is not satisfied. In the maritime domain, e.g., we may say that a vessel is performing a 'trawling' fishing activity if the vessel started performing sharp turns in a fishing area a short time ago, and has not left the fishing area since.

There are several frameworks for temporal pattern matching that employ automata. CORE, e.g., is an automata-based system with highly efficient reasoning algorithms (Bucchi et al. 2022). CORE, however, does not support relational concepts, and thus cannot express a 'meeting' activity between two persons. Furthermore, automata-based systems commonly model time implicitly (Giatrakos et al. 2020), complicating the representation of the law of inertia. There are also logic-based frameworks. MeTeoR extends a fragment of DatalogMTL with windows, but does not support negation (Walega et al. 2023). LARS (Beck et al. 2018) is a temporal specification language with several implementations (Beck et al. 2017; Bazoobandi et al. 2017; Eiter et al. 2019). LARS-based engines cannot express durative composite activity definitions, such as 'trawling'.

The Event Calculus is a logic programming formalism for representing and reasoning about events and their effects over time (Kowalski and Sergot 1986); it includes instantaneous events and time-varying properties called 'fluents'. The Event Calculus supports durative and relational situations of interest, and a built-in representation of inertia. Several Event Calculus-based frameworks are found in the literature (Chittaro and Montanari 1996; Paschke and Bichler 2008; Chesani et al. 2013; Kafali et al. 2017; Baumgartner 2021; Arias et al. 2022). jREC, e.g., is a 'reactive' Event Calculus dialect that supports data streams (Montali et al. 2013; Falcionelli et al. 2019).

The Run-Time Event Calculus (RTEC) extends the Event Calculus with optimisation techniques for data streams, such windowing, caching and indexing (Artikis et al. 2015; Mantenoglou et al. 2022, 2023), and has proven highly efficient in real applications (Tsilionis et al. 2022). In RTEC, it is possible to represent a situation of interest as a 'simple' or a 'statically determined' fluent. Simple fluents are defined in terms of their initiations and terminations, and the law of inertia. Statically determined fluents are defined as Boolean combinations of other fluents. While a simple fluent representation for such a Boolean combination is possible, it is exponentially larger and more expensive to reason with than the corresponding statically determined fluent definition. Therefore, whenever possible, a situation of interest should be defined as a statically determined fluent.

Unfortunately, the knowledge engineer may not always employ the most efficient definition of a fluent. For instance, the engineer may be working on a knowledge base that was developed for an Event Calculus-based framework that does not feature statically determined fluents. In a legal contract verification framework (Sharifi et al. 2020; Parvizimosaed

et al. 2022), e.g., the Event Calculus representation for the rights-holder of a contract can be optimised using a statically determined fluent definition. As another example, in a distributed authorisation policy management framework (Zahoor et al. 2022, 2023), the Event Calculus representations for policy conflicts and redundant policies can be re-written as statically determined fluents. Moreover, the engineer may detect only a portion of the composite activities that can be defined as statically determined fluents, resulting in inefficiencies when reasoning over the simple fluents that could have been written as statically determined ones. To address this issue, we propose a compiler that optimises temporal specifications in RTEC.

Our contributions are the following. First, we formally identify the class of simple fluent definitions that can be translated into statically determined fluent definitions. Second, we provide a compiler for the translation and prove its correctness. Third, we present a reproducible empirical evaluation on knowledge bases and data streams of real applications, i.e., composite event recognition, legal contract verification, clinical guideline monitoring and distributed authorisation policy management. Our compiler reduced the size of these knowledge bases by orders of magnitude, leading to significant reasoning efficiency gains. The proofs of all propositions of the paper and the resources for reproducing our experiments are publicly available[1].

The Run-Time Event Calculus

Representation. The language of RTEC is many-sorted, including sorts for representing time, instantaneous events and fluents. RTEC employs a linear time-line with non-negative integer time-points. A 'fluent-value pair' (FVP) $F=V$ denotes that fluent F has value V. $\mathsf{happensAt}(E, T)$ signifies that event E occurs at time-point T. $\mathsf{initiatedAt}(F = V, T)$ (resp. $\mathsf{terminatedAt}(F = V, T)$) expresses that a time period during which a fluent F has the value V continuously is initiated (terminated) at T. $\mathsf{holdsAt}(F = V, T)$ states that F has value V at T, while $\mathsf{holdsFor}(F = V, I)$ expresses that $F=V$ holds continuously in the intervals included in list I.

A formalisation of the temporal specifications of a domain in RTEC is called *event description*.

Definition 1 (Event Description)**.** An event description is a set of:

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

RTEC features two types of FVPs: 'simple' and 'statically determined'. A simple FVP $F = V$ is defined via a set of $\mathsf{initiatedAt}(F = V, T)$ and $\mathsf{terminatedAt}(F = V, T)$ rules.

Example 1 ('Meeting' as a Simple FVP)**.** In human activity recognition, the task is to detect composite human activities of interest, such as illegal, suspicious and violent activities, based on streams containing symbolic representations of video feeds. We may use the fluent $meeting(P_1, P_2)$ to express that people P_1 and P_2 are having a meeting, while the value of $meeting(P_1, P_2)$ denotes the stage of the meeting. The simple FVP $meeting(P_1, P_2) = interacting$ can be specified with the following rules:

$$\begin{aligned}&\mathsf{initiatedAt}(meeting(P_1, P_2) = interacting,\ T) \leftarrow\\ &\quad \mathsf{happensAt}(\mathsf{start}(active(P_1) = \mathsf{true}),\ T),\\ &\quad \mathsf{holdsAt}(close(P_1, P_2) = \mathsf{true},\ T),\\ &\quad \text{not } \mathsf{happensAt}(\mathsf{end}(close(P_1, P_2) = \mathsf{true}),\ T).\end{aligned} \tag{1}$$

$$\begin{aligned}&\mathsf{initiatedAt}(meeting(P_1, P_2) = interacting,\ T) \leftarrow\\ &\quad \mathsf{happensAt}(\mathsf{start}(close(P_1, P_2) = \mathsf{true}),\ T),\\ &\quad \mathsf{holdsAt}(active(P_1) = \mathsf{true},\ T),\\ &\quad \text{not } \mathsf{happensAt}(\mathsf{end}(active(P_1) = \mathsf{true}),\ T).\end{aligned} \tag{2}$$

$$\begin{aligned}&\mathsf{initiatedAt}(meeting(P_1, P_2) = interacting,\ T) \leftarrow\\ &\quad \mathsf{happensAt}(\mathsf{start}(active(P_1) = \mathsf{true}),\ T),\\ &\quad \mathsf{happensAt}(\mathsf{start}(close(P_1, P_2) = \mathsf{true}),\ T).\end{aligned} \tag{3}$$

$$\begin{aligned}&\mathsf{terminatedAt}(meeting(P_1, P_2) = interacting,\ T) \leftarrow\\ &\quad \mathsf{happensAt}(\mathsf{end}(active(P_1) = \mathsf{true}),\ T).\end{aligned} \tag{4}$$

$$\begin{aligned}&\mathsf{terminatedAt}(meeting(P_1, P_2) = interacting,\ T) \leftarrow\\ &\quad \mathsf{happensAt}(\mathsf{end}(close(P_1, P_2) = \mathsf{true}),\ T).\end{aligned} \tag{5}$$

According to rules (1)–(3), P_1 and P_2 start interacting when P_1 starts being 'active', while P_1 and P_2 remain close to each other (rule (1)), P_1 and P_2 come close to each other, while P_1 continues being active (rule (2)), or P_1 starts being active, and P_1 and P_2 start being close to each other at the same time (rule (3)). According to rules (4)–(5), P_1 and P_2 stop interacting when P_1 stops being active (rule (4)), or when P_1 and P_2 stop being close to each other (rule (5)). ♢

The auxiliary start and end events allow for succinct event descriptions and reasoning optimisations. For a simple FVP $F = V$, these events are defined as follows:

$$\begin{aligned}&\mathsf{happensAt}(\mathsf{start}(F = V),\ T) \leftarrow\\ &\quad \mathsf{initiatedAt}(F = V,\ T), \text{not } \mathsf{holdsAt}(F = V,\ T).\end{aligned} \tag{6}$$

$$\begin{aligned}&\mathsf{happensAt}(\mathsf{end}(F = V),\ T) \leftarrow\\ &\quad \mathsf{terminatedAt}(F = V,\ T), \mathsf{holdsAt}(F = V,\ T).\end{aligned} \tag{7}$$

Rule (6) states that $\mathsf{start}(F = V)$ takes place at time-point T if $F = V$ is initiated at T and $F = V$ does not hold at T, i.e., $\mathsf{start}(F = V)$ marks the initiations of $F = V$ that bring about a new time period where $F = V$ holds continuously. Similarly, based on rule (7), $\mathsf{end}(F = V)$ marks the terminations T of $F = V$ for which $F = V$ holds at T.

Definition 2 (Syntax of Rules Defining Simple FVPs)**.** Consider a simple FVP $F = V$. The $\mathsf{initiatedAt}(F = V, T)$ rules of the event description have the following syntax:

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

The first body literal of an initiatedAt rule is a positive happensAt predicate; this is followed by a possibly empty set, denoted by '[[ ]]', of positive/negative happensAt and holdsAt predicates. 'not' expresses negation-by-failure (Clark

[1] https://github.com/periklismant/aaai2025_supplementary

1977), while '[not]' denotes that 'not' is optional. E_i, where $i \in [1, n]$, denotes an event of the domain, or an auxiliary event start$(F' = V')$ or end$(F' = V')$, where $F' = V'$ is an FVP. All (head and body) predicates are evaluated on the same time-point T. The bodies of terminatedAt$(F = V, T)$ rules have the same form. ■

The definition of a simple FVP $F = V$ is a pair of sets (R^s, R^e); set R^s (resp. R^e) contains rules with head initiatedAt$(F = V, T)$ (terminatedAt$(F = V, T)$). (R^s_m, R^e_m), e.g., where R^s_m (R^e_m) contains rules (1)–(3) (rules (4)–(5)), is a definition of $meeting(P_1, P_2) = interacting$.

A statically determined FVP $F = V$ is defined via a rule with head holdsFor$(F = V, I)$.

Example 2 ('Meeting' as a Statically Determined FVP)**.** We may define $meeting(P_1, P_2) = interacting$ as a statically determined FVP using the following rule r_m:

$$\begin{aligned} &\text{holdsFor}(meeting(P_1, P_2) = interacting,\ I) \leftarrow \\ &\quad \text{holdsFor}(active(P_1) = \text{true},\ I_a), \\ &\quad \text{holdsFor}(close(P_1, P_2) = \text{true},\ I_c), \\ &\quad \text{intersect\_all}([I_a, I_c], I). \end{aligned} \tag{8}$$

intersect_all$([I_a, I_c], I)$ is an interval manipulation construct, computing the list of maximal intervals I as the intersection of all maximal intervals in lists I_a and I_c. r_m states that $meeting(P_1, P_2) = interacting$ holds in the intervals where $active(P_1) = \text{true}$ and $close(P_1, P_2) = \text{true}$ hold. ♢

Definition 3 (Syntax of Rules Defining Statically Determined FVPs)**.** The definition of statically determined FVP $F = V$ is a rule that has the following syntax:

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

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

union_all(L, I) (resp. intersect_all(L, I)) computes the list of maximal intervals I as the union (intersection) of all lists of maximal intervals of list L. relative_complement_all(I', L, I) computes the list of maximal intervals I by removing from the maximal intervals of list I' all interval segments included in an interval of some list in L.

The interval manipulation constructs of RTEC support the following type of definition: for all time-points T, $F = V$ holds at T if and only if some Boolean combination of FVPs holds at T. To aid the presentation, we often represent such Boolean combinations with 'Boolean FVP definitions', i.e., DNF formulas with literals that correspond to FVPs; these definitions are not in the language of RTEC. A Boolean FVP definition p_m of $meeting(P_1, P_2) = interacting$, e.g., is:

$$active(P_1) = \text{true} \wedge close(P_1, P_2) = \text{true} \tag{9}$$

In the following sections, we will show that p_m, r_m, i.e., rule (8), and (R^s_m, R^e_m), i.e., rules (1)–(5), are equivalent definitions of $meeting(P_1, P_2) = interacting$.

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 each FVP holds continuously. For a statically determined FVP $F = V$ with definition r, RTEC computes holdsFor$(F = V, I)$ by evaluating the conditions of rule r. For a simple FVP $F = V$ with definition (R^s, R^e), RTEC operates as follows. First, RTEC computes the initiations and the terminations of $F = V$, by evaluating the rules in sets R^s and R^e, respectively. Next, RTEC 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. RTEC may then derive holdsAt$(F = V, T)$ by checking whether T belongs to one of the maximal intervals of $F = V$.

From Simple to Statically Determined FVPs

In RTEC, a composite activity expressing a Boolean combination of other activities may be modelled as a simple or a statically determined FVP. The size of the resulting simple FVP definition, however, is exponential to the number of literals in the Boolean definition. The Boolean FVP definition of $meeting(P_1, P_2) = interacting$ in formula (9), e.g., is a conjunction of 2 FVPs, i.e., $active(P_1) = \text{true}$ and $close(P_1, P_2) = \text{true}$. In order to specify the initiations of $meeting(P_1, P_2) = interacting$ in rules (1)–(3), we had to explicitly declare the possible orders in which FVPs $active(P_1) = \text{true}$ and $close(P_1, P_2) = \text{true}$ may come into effect, using one rule for each such order, leading to 3 rules. In contrast, the equivalent statically determined FVP definition is *1* rule with 3 conditions (rule (8)). To express a conjunction of n FVPs with a simple FVP definition, we need $2^n - 1$ initiation rules, i.e., one rule for each subset of the FVPs in the conjunction that may start simultaneously; the equivalent statically determined FVP definition is *1* rule with $n+1$ conditions. Moreover, according to the complexity analysis of RTEC (Artikis et al. 2015), the worst-case cost of deriving the maximal intervals of a statically determined FVP is considerably lower than the cost of maximal interval computation for a simple FVP. These claims are further supported by our empirical evaluation, where we demonstrate that, in real applications, the use of statically determined FVPs leads to much more concise event descriptions and significantly more efficient reasoning than the use of simple FVPs. Therefore, whenever possible, a composite activity should be defined as a statically determined FVP.

Previous work assumed that the event description developer was able to optimise an event description, by, e.g., identifying whether a composite activity can be expressed as a statically determined FVP (Artikis et al. 2015). Unfortunately, such as an assumption rarely holds in practice. Towards developing a compiler that automates event description development and addresses this issue, we formally iden-

tify the class of simple FVP definitions that may be translated into equivalent statically determined FVP definitions. We use the following notion of FVP definition equivalence:

Definition 4 (Equivalent FVP Definitions)**.** Two definitions, D_1 and D_2, for FVP $F{=}V$ are equivalent, if D_1 implies $\mathsf{holdsAt}(F = V, T)$ iff D_2 implies $\mathsf{holdsAt}(F = V, T)$. ■

Proposition 1 (Equivalence between Statically Determined and Boolean FVP Definitions)**.** For each statically determined FVP definition, there is an equivalent Boolean FVP definition, and vice versa. ♦

The statically determined FVP definition in rule (8), e.g., is equivalent with the Boolean FVP definition in formula (9).

Proposition 1 states that every Boolean FVP definition can be translated into a statically determined FVP definition, and vice versa. Therefore, a simple FVP definition (R^s, R^e) is *translatable* into a statically determined FVP definition iff there is a Boolean FVP definition that is equivalent with (R^s, R^e). We outline, as an example, a proof demonstrating that simple FVP definition (R^s_m, R^e_m) is translatable. Then, we present the class of translatable simple FVP definitions.

Equivalence Proof for Running Example

We prove that the simple FVP definition (R^s_m, R^e_m), containing rules (1)–(5), and the Boolean FVP definition p_m in formula (9) are equivalent, i.e., they compute the same holdsAt atoms for FVP $meeting(P_1, P_2) = interacting$. p_m implies $\mathsf{holdsAt}(meeting(P_1, P_2) = interacting, T)$ iff both $active(P_1) = \mathsf{true}$ and $close(P_1, P_2) = \mathsf{true}$ hold at time-point T. (R^s_m, R^e_m) implies the same $\mathsf{holdsAt}(meeting(P_1, P_2) = interacting, T)$ atoms as the program comprising rules (1)–(5) and the following formulation of the law of inertia:

$$\mathsf{holdsAt}(F = V,\ T) \leftarrow \mathsf{initiatedAt}(F = V,\ T{-}1). \quad (10)$$

$$\mathsf{holdsAt}(F = V, T) \leftarrow \mathsf{holdsAt}(F = V, T{-}1),\ \mathsf{not\ terminatedAt}(F = V, T{-}1). \quad (11)$$

Rule (10) states that $F = V$ holds at time-point T if $F = V$ is initiated at the previous time-point $T{-}1$. Rule (11) expresses that $F = V$ holds at T if $F = V$ holds at $T{-}1$ and $F{=}V$ is not terminated at $T{-}1$.

We prove that (R^s_m, R^e_m) and p_m are equivalent with an inductive proof on time-point T. Below, we outline the inductive step for time-point T (equivalences (12)–(16)), assuming that (R^s_m, R^e_m) and p_m imply the same holdsAt atoms at time-point $T{-}1$. The base of the induction is presented in the technical appendix[1].

$$(R^s_m, R^e_m) \models \mathsf{holdsAt}(meeting(P_1, P_2) = interacting, T)$$

$$\leftrightarrow (\mathsf{initiatedAt}(meeting(P_1, P_2) = interacting, T{-}1) \quad (12)$$
$$\vee (\mathsf{holdsAt}(meeting(P_1, P_2) = interacting, T{-}1) \wedge$$
$$\neg \mathsf{terminatedAt}(meeting(P_1, P_2) = interacting, T{-}1)))$$

$$\leftrightarrow (\mathsf{b}_{(1)} @\, T{-}1 \vee \mathsf{b}_{(2)} @\, T{-}1 \vee \mathsf{b}_{(3)} @\, T{-}1 \vee \quad (13)$$
$$(\mathsf{holdsAt}(active(P_1) = \mathsf{true}, T{-}1) \wedge \neg \mathsf{b}_{(4)} @\, T{-}1 \wedge$$
$$\mathsf{holdsAt}(close(P_1, P_2) = \mathsf{true}, T{-}1) \wedge \neg \mathsf{b}_{(5)} @\, T{-}1))$$

$$\leftrightarrow ((\mathsf{happensAt}(\mathsf{start}(active(P_1) = \mathsf{true}), T{-}1) \vee \quad (14)$$
$$(\mathsf{holdsAt}(active(P_1) = \mathsf{true}, T{-}1) \wedge$$
$$\neg \mathsf{happensAt}(\mathsf{end}(active(P_1) = \mathsf{true}), T{-}1))) \wedge$$
$$(\mathsf{happensAt}(\mathsf{start}(close(P_1, P_2) = \mathsf{true}), T{-}1) \vee$$
$$(\mathsf{holdsAt}(close(P_1, P_2) = \mathsf{true}, T{-}1) \wedge$$
$$\neg \mathsf{happensAt}(\mathsf{end}(close(P_1, P_2) = \mathsf{true}), T{-}1))))$$

$$\leftrightarrow (\mathsf{holdsAt}(active(P_1) = \mathsf{true}, T) \wedge \quad (15)$$
$$\mathsf{holdsAt}(close(P_1, P_2) = \mathsf{true}, T))$$

$$\leftrightarrow p_m \models \mathsf{holdsAt}(meeting(P_1, P_2) = interacting, T) \quad (16)$$

'$D \models l$' expresses that l is true in all models of D, while '$\mathsf{b}_{(i)} @\, T{-}1$' denotes (the conjunction of) the body condition(s) of rule (i), having $T{-}1$ as their time argument. Equivalence (12) expresses the law of inertia, i.e., (R^s_m, R^e_m) implies that $meeting(P_1, P_2) = interacting$ holds at time-point T iff it is initiated at time-point $T{-}1$ or it holds and is not terminated at $T{-}1$. Equivalence (13) includes a substitution of $\mathsf{initiatedAt}(meeting(P_1, P_2) = interacting, T{-}1)$ and $\mathsf{terminatedAt}(meeting(P_1, P_2) = interacting, T{-}1)$ using the bodies of rules (1)–(3) and the bodies of rules (4)–(5), respectively. Moreover, based on our inductive assumption, we have substituted $\mathsf{holdsAt}(meeting(P_1, P_2) = interacting, T{-}1)$ with the conjunction of $\mathsf{holdsAt}(active(P_1) = \mathsf{true}, T{-}1)$ and $\mathsf{holdsAt}(close(P_1, P_2) = \mathsf{true}, T{-}1)$, following p_m. Equivalence (14) distributes $\wedge$ over $\vee$, while equivalence (15) expresses the law of inertia for FVPs $active(P_1) = \mathsf{true}$ and $close(P_1, P_2) = \mathsf{true}$. Equivalences (14)–(15) are explained thoroughly in the technical appendix[1]. Equivalence (16) expresses p_m, i.e., $meeting(P_1, P_2) = interacting$ holds iff both $active(P_1) = \mathsf{true}$ and $close(P_1, P_2) = \mathsf{true}$ hold.

We have shown that definitions (R^s_m, R^e_m) and p_m imply the same $\mathsf{holdsAt}(meeting(P_1, P_2) = interacting, T)$ atoms. Therefore, definitions (R^s_m, R^e_m) and p_m are equivalent. Since p_m is equivalent with statically determined FVP definition r_m (rule (8)), it follows that definition (R^s_m, R^e_m) is equivalent with, and translatable into, definition r_m.

The Class of Translatable Simple FVP Definitions

We identify symmetries that are satisfied by a simple FVP definition iff it is translatable.

Inertial Condition Symmetry. Before presenting the first symmetry, we define the *inertial condition set* of an FVP.

Definition 5 (Inertial Condition Set)**.** Consider an FVP literal l. If l is $F{=}V$, then the inertial condition set IC_l of l contains the following conditions:

- '$\mathsf{holdsAt}(F = V, T), \mathsf{not\ happensAt}(\mathsf{end}(F = V), T)$'.
- '$\mathsf{happensAt}(\mathsf{start}(F = V), T)$'.

If l is $\neg(F = V)$, then IC_l contains:

- '$\mathsf{not\ holdsAt}(F = V, T), \mathsf{not\ happensAt}(\mathsf{start}(F = V), T)$'.
- '$\mathsf{happensAt}(\mathsf{end}(F = V), T)$'. ■

A simple FVP definition (R^s, R^e) is translatable only if R^s and R^e are *inertial condition symmetric* rule-sets.

Definition 6 (Inertial Condition Symmetric Rule-Set)**.** A rule-set R is inertial condition symmetric if, $\forall r \in R$, for each inertial condition c in r, with FVP u, where c is not the only positive happensAt condition in r, $\exists r' \in R$, such that r' differs from r only by having a condition c' instead of c, where $c' \neq c$ and $c' \in IC_u$. ■

Consider, e.g., rule-set R^s_m, i.e., rules (1)–(3). Rules (1) and (3), and rules (2) and (3), differ only in the inertial condition of FVP $close(P_1, P_2) = \text{true}$ and $active(P_1) = \text{true}$, respectively. Thus, R^s_m is inertial condition symmetric. R^e_m is also inertial condition symmetric; the rules in R^e_m contain one inertial condition each, which is a positive happensAt.

As a counter-example, consider set R^{s-}_m with rules (2) and (3). R^{s-}_m is not inertial condition symmetric because there is no rule in R^{s-}_m for one of the orders in which the FVPs in its inertial conditions may come about, i.e., $active(P_1) = \text{true}$ starting to hold after $close(P_1, P_2) = \text{true}$ (Definition 6).

An inertial condition symmetric rule-set is *minimal* if it cannot be partitioned into more than one inertial condition symmetric rule-sets. R^s_m, e.g., is minimal, while R^e_m is not minimal. A partition of an inertial condition symmetric rule-set is *complete* if it includes only minimal inertial condition symmetric sets. The partition of R^e_m, e.g., into sets R^{e1}_m and R^{e2}_m, with rule (4) and rule (5), respectively, is complete.

Guard Condition Symmetry. A translatable simple FVP definition may optionally contain 'guard conditions'; their role is to prevent the computation of redundant FVP initiations/terminations. Consider, e.g., definition (R^s_m, R'^e_m), where R'^e_m contains the following rules:

$$\text{terminatedAt}(meeting(P_1, P_2) = interacting,\ T) \leftarrow \text{happensAt}(\text{end}(active(P_1) = \text{true}),\ T), \text{holdsAt}(close(P_1, P_2) = \text{true},\ T). \quad (17)$$

$$\text{terminatedAt}(meeting(P_1, P_2) = interacting,\ T) \leftarrow \text{happensAt}(\text{end}(close(P_1, P_2) = \text{true}),\ T), \text{holdsAt}(active(P_1) = \text{true},\ T). \quad (18)$$

We constructed rule (17), e.g., by adding guard condition $\text{holdsAt}(close(P_1, P_2) = \text{true}, T)$ in rule (4), i.e., the rule in set R^{e1}_m of the complete partition of R^e_m. This condition states that $close(P_1, P_2) = \text{true}$ needs to be in effect when a rule in R^{e1}_m fires; otherwise, the termination would be redundant, because $meeting(P_1, P_2) = interacting$ would have been terminated already by a rule in the minimal inertial condition symmetric rule-set R^{e2}_m of R^e_m, due to condition $\text{happensAt}(\text{end}(close(P_1, P_2) = \text{true}), T)$ of R^{e2}_m. We define the *guard condition set* of a rule-set as follows:

Definition 7 (Guard Condition Set)**.** Consider a minimal inertial condition symmetric rule-set R and set U, containing the FVP literals corresponding to the inertial conditions in R. The guard condition set of R is: $G = \bigcup_{l \in U} \{c\}$, where c is '$\text{not holdsAt}(F = V, T)$', if l is $F = V$, and '$\text{holdsAt}(F = V, T)$', if l is $\neg(F = V)$. ■

A simple FVP definition (R^s, R^e) is translatable only if R^s and R^e are *guard condition symmetric* rule-sets.

Definition 8 (Guard Condition Symmetric Rule-Set)**.** Consider a rule-set R. R is guard condition symmetric if it is inertial condition symmetric, $G^1, G^2, \ldots, G^n$ are the guard condition sets of the sets $R^1, R^2, \ldots, R^n$ in the complete partition of R, and $\forall i \in \{1, \ldots, n\}$, $\forall r \in R^i$, for each guard condition c in r, where $c \in G^j$, $j \neq i$, it holds that $\forall c' \in G^j$, where $c' \neq c$, $\exists r' \in R$, such that r' differs from r only by having condition c' instead of c. ■

R'^e_m, e.g., is inertial condition symmetric and its complete partition consists of sets R'^{e1}_m and R'^{e2}_m, containing rule (17) and rule (18), respectively. The guard condition sets G'^{e1}_m and G'^{e2}_m of R'^{e1}_m and R'^{e2}_m contain one condition each, i.e., $\text{holdsAt}(close(P_1, P_2) = \text{true}, T)$ and $\text{holdsAt}(active(P_1) = \text{true}, T)$, resp. Thus, following Definition 8, R'^e_m is guard condition symmetric. Moreover, R^s_m and R^e_m are guard condition symmetric because they are inertial condition symmetric and contain no guard conditions.

Boolean Representation Symmetry. Equivalence (14) stated that the bodies of the initiation and termination rules in (R^s_m, R^e_m) need to include the conditions expressing the law of inertia for the conjunction of $active(P_1) = \text{true}$ and $close(P_1, P_2) = \text{true}$, and no other inertial conditions. This is only possible when sets R^s_m and R^e_m are complementary in terms of inertial conditions. Towards specifying this type of relation, we define the *Boolean representation* of an inertial condition symmetric rule-set.

Definition 9 (Boolean Representation of Inertial Condition Symmetric Rule-Set)**.** Consider an inertial condition symmetric rule-set R, where $R^1, R^2, \ldots, R^n$ is a complete partition of R. Set U^i, where $1 \leq i \leq n$, contains the FVP literals corresponding to the inertial conditions in R^i. The Boolean representation of R is: $p = \bigvee_{1 \leq i \leq n} \wedge_{l \in U^i} l$. ■

The Boolean representation of the minimal inertial condition symmetric rule-set R^s_m, e.g., is '$active(P_1) = \text{true} \wedge close(P_1, P_2) = \text{true}$', while the Boolean representation of rule-set R^e_m, with complete partition R^{e1}_m and R^{e2}_m, is '$\neg(active(P_1) = \text{true}) \vee \neg(close(P_1, P_2) = \text{true})$'.

A simple FVP definition (R^s, R^e) is translatable only if it is *Boolean representation symmetric*.

Definition 10 (Boolean Representation Symmetric Simple FVP Definition)**.** A simple FVP definition (R^s, R^e) is Boolean representation symmetric if sets R^s and R^e are inertial condition symmetric and the Boolean representations p and $\overline{p}$ of R^s and R^e are complementary. ■

(R^s_m, R^e_m) is Boolean representation symmetric as the Boolean representations of R^s_m and R^e_m are complementary.

We formally identify the class of translatable simple FVP definitions with the following proposition.

Proposition 2 (Translatable Simple FVP Definition)**.** A simple FVP definition (R^s, R^e) is translatable iff:
1. R^s and R^e are guard condition symmetric, and
2. (R^s, R^e) is Boolean representation symmetric.

When (R^s, R^e) is translatable, it is equivalent with the definition induced by the Boolean representation of R^s. ♦

We have shown, e.g., that sets R^s_m and R^e_m are guard condition symmetric, and that (R^s_m, R^e_m) is Boolean representation symmetric. Thus, based on Proposition 2, simple FVP definition (R^s_m, R^e_m) is translatable. Moreover, (R^s_m, R^e_m) is equivalent with the Boolean FVP definition induced by the Boolean representation of R^s_m, which is p_m (formula (9)).

Compiler

We developed an open-source compiler[1] that optimises an event description by replacing all translatable simple FVP definitions with their equivalent statically determined FVP definitions. Our empirical analysis, presented in the following section, demonstrates that the compiled event descrip-

Algorithm 1: $compiler(R^s, R^e)$

1: **if not** $guardSymm(R^s)$ **or not** $guardSymm(R^e)$ **then**
2: **return** false
3: $p \leftarrow BoolRepr(R^s)$, $\overline{p} \leftarrow BoolRepr(R^e)$
4: **if not** $complementary(p, \overline{p})$ **then return** false
5: $h_r \leftarrow$ holdsFor$(F = V, I)$, $b_r \leftarrow [\,]$
6: **for each** disjunct d_i **in** p **do**
7: **for each** literal l_{ij} **in** d_i **do**
8: **if** l_{ij} is $F_{ij} = V_{ij}$ **then**
9: b_r.add(holdsFor$(F_{ij} = V_{ij}, I_{ij})$)
10: **else** ▷ l_{ij} is $\neg(F_{ij} = V_{ij})$.
11: b_r.add(holdsFor$(F_{ij} = V_{ij}, I'_{ij})$)
12: b_r.add(relative_complement_all$([i_w], I'_{ij}, I_{ij})$)
13: **if** $|d_i|>1$ **then** b_r.add(intersect_all$([I_{i1}, \ldots, I_{in_i}], I_i)$)
14: **if** $|p|>1$ **then** b_r.add(union_all$([I_1, \ldots, I_m], I)$)
15: **return** r

tions are considerably more succinct, and lead to significantly more efficient reasoning. Algorithm 1 outlines the steps of the compiler. For each input definition (R^s, R^e) of a simple FVP $F=V$, the compiler works as follows. First, it identifies whether (R^s, R^e) is translatable (lines 1–4). Second, if (R^s, R^e) is translatable, it rewrites (R^s, R^e) as an equivalent statically determined FVP definition (lines 5–14).

We check whether (R^s, R^e) is translatable by examining the symmetry conditions of Proposition 2. We start by checking whether sets R^s and R^e are guard condition symmetric (lines 1–2). To do this, we first need to verify that R^s and R^e are inertial condition symmetric (see Definition 8). For R^s we proceed as follows. For each rule r^{si} in R^s, we identify the FVPs in the inertial conditions of r^{si} and construct set R^{si}, containing one rule for each combination of the inertial conditions of these FVPs with at least one positive happensAt condition. If $\forall r^{si} \in R^s$, we have $R^{si} \subseteq R^s$, then R^s is inertial condition symmetric (Definition 6). The disjoint sets $R^{s1}, \ldots, R^{sm}$ are a complete partition of R^s. Then, we construct the guard condition sets $G^{s1}, \ldots, G^{sm}$ of $R^{s1}, \ldots, R^{sm}$ and, for each rule-set R^{si} and guard condition set G^{sj}, where $j \neq i$, we verify that, if there is a rule r in R^{si} with a guard condition $c \in G^{sj}$, then R^{si} contains, for each $c' \in G^{sj}$, one rule that differs from r only by having guard condition c' instead of c (Definition 8). The cost of these procedures is quadratic to the number of rule conditions in R^s. Afterwards, we construct Boolean representations p and $\overline{p}$ of R^s and R^e based their complete partitions $R^{s1}, \ldots, R^{sm}$ and $R^{e1}, \ldots, R^{em'}$, following Definition 9 (line 3), and check whether p and $\overline{p}$ are complementary (line 4). If they are, then (R^s, R^e) is Boolean representation symmetric (Definition 10). If the above symmetries are satisfied, then (R^s, R^e) is equivalent with Boolean FVP definition p, i.e., the Boolean representation of R^s.

Algorithm 1 translates p, i.e., the Boolean FVP definition that is equivalent with (R^s, R^e), into statically determined FVP definition r, with head holdsFor$(F = V, I)$, in lines 5–14. For each disjunct d_i in p, we process each literal l_{ij} in d_i (lines 6–7). If l_{ij} is $F_{ij} = V_{ij}$, then we add condition holdsFor$(F_{ij} = V_{ij}, I_{ij})$ in the body b_r of r (lines 8–9). List I_{ij} contains the maximal intervals during which $F_{ij} = V_{ij}$ holds continuously. Otherwise, if l_{ij} is $\neg(F_{ij} = V_{ij})$, then we are interested in the maximal intervals during which $F_{ij} = V_{ij}$ does not hold. We express the list I_{ij} that contains these intervals with conditions holdsFor$(F_{ij} = V_{ij}, I'_{ij})$ and relative_complement_all$([i_w], I'_{ij}, I_{ij})$, where interval i_w expresses the window, i.e., the bounded portion of the stream currently being processed by RTEC (lines 10–12). If d_i contains more than one FVP literal, denoted by '$|d_i|>1$', we add intersect_all$([I_{i1}, \ldots I_{in_i}], I_i)$ in b_r (line 13). This way, we express the list of maximal intervals I_i during which all FVP literals in d_i hold. Finally, if p contains more than one disjunct, denoted by '$|p|>1$', we add union_all$([I_1, \ldots, I_m], I)$ in b_r (line 14); list I contains the maximal intervals during which at least one of the disjuncts of p is satisfied. For this reason, r implies that $F=V$ holds at some time-point T iff p is satisfied at T, i.e., definitions r and p are equivalent. Thus, r is equivalent with the input simple FVP definition (R^s, R^e).

Example 3 (Compiling a Simple FVP Definition of 'Meeting')**.** Simple FVP definition (R^s_m, R^e_m) is translatable into Boolean FVP definition p_m (formula (9)). Our compiler verifies this in lines 1–4 of Algorithm 1, and then proceeds with the translation of p_m into a rule r, with head holdsFor$(meeting(P_1, P_2) = interacting, I)$ (line 5). p_m has one disjunct, d_1, with two literals, l_{11}: $active(P_1) =$ true and l_{12}: $close(P_1, P_2) =$ true. Based on these literals, we add holdsFor$(active(P_1) =$ true$, I_{11})$ and holdsFor$(close(P_1, P_2) =$ true$, I_{12})$ in the body b_r of r (lines 8–9). Since d_1 contains two FVP literals, we need to express the maximal intervals during which both these literals hold. To do this, we add intersect_all$([I_{11}, I_{12}], I)$ in b_r (line 13). p_m contains one disjunct, and thus we do not add union_all (line 14). The resulting rule r is the same as statically determined FVP definition r_m (rule (8)). ◊

Proposition 3 (Compiler Correctness)**.** Given a simple FVP definition (R^s, R^e), if (R^s, R^e) is translatable, then Algorithm 1 returns a statically determined FVP definition that is equivalent with (R^s, R^e). Otherwise, it returns false. ♦

Experimental Analysis

Experimental Setup. We evaluated our compiler on handcrafted Event Calculus formalisations for composite event recognition (CER), legal contract verification, clinical guideline monitoring and distributed authorisation policy management. In all applications, the input event descriptions included only simple FVP definitions. Our compiler optimised the event descriptions, i.e., identified the translatable FVP definitions and expressed them as statically determined FVP definitions. Then, we compared the size of the input and compiled event descriptions, and the efficiency of RTEC when reasoning over each event description.

Concerning CER, we tested our compiler on three tasks: human activity recognition, maritime situational awareness and city transport management. In *human activity recognition*, the input streams were symbolic representations of video feeds, including events that denote simple activities

performed by one person and identified on individual video frames, such as 'active', as well as the coordinates and the orientation of the tracked people. The task was to compute the maximal intervals of 'meeting', 'moving together', 'fighting' and 'leaving object' activities. We used CAVIAR[2], a benchmark activity recognition dataset. For *maritime situational awareness*, we used streams of events that were derived from Automatic Identification System (AIS) signals, containing information about vessels' location, speed and heading. The task was to compute the maximal intervals of various types of dangerous, suspicious or illegal vessel activities, such as a ship-to-ship transfer of goods in the open sea (Pitsikalis et al. 2019). We used a publicly available dataset[3], containing 18M AIS signals, emitted by 5K vessels sailing around the port of Brest, France, between October 2015–March 2016. In *city transport management*, we used streams of events concerning changes in the position, acceleration, in-vehicle temperature, noise level and passenger density of public transport vehicles. The task was to compute the maximal intervals of composite activities related to public transport vehicle punctuality, driving style and quality, and passenger and driver comfort and satisfaction. We used real data, collected from the public transport vehicles in Helsinki, Finland, in November 2011 (Artikis et al. 2015).

We also evaluated our compiler on Event Calculus formalisations for tasks other than CER. For *legal contract verification*, we used an event description specifying the obligations and the powers of the parties in a contract (Sharifi et al. 2020; Parvizimosaed et al. 2022), while, for *clinical guideline monitoring*, we used an event description defining scenarios of non-conformance with clinical guidelines (Bottrighi et al. 2011; Bragaglia et al. 2012). Moreover, we employed event descriptions specifying *authorisation policy conflicts* in multi-cloud environments (Zahoor et al. 2022), and *redundant authorisation policies* in Kubernetes (Zahoor et al. 2023). Unfortunately, no datasets are available for these applications, and thus we could only compare the size of the input and compiled event descriptions.

Our experiments are *reproducible*; our compiler, the input and compiled event descriptions, and the datasets are publicly available[1]. RTEC[4] operated on SWI-8.4 Prolog on a PC with Ubuntu 22, Ryzen 7 5700U and 16GB RAM.

Experimental Results. The input event descriptions include only simple FVPs; $\mathcal{E}^i_h$, $\mathcal{E}^i_m$, $\mathcal{E}^i_t$, $\mathcal{E}^i_l$, $\mathcal{E}^i_g$, $\mathcal{E}^i_c$ and $\mathcal{E}^i_r$ are the event descriptions for **h**uman activity recognition, **m**aritime situational awareness, city **t**ransport management, **l**egal contract verification, clinical **g**uideline monitoring, and the identification of authorisation policy **c**onflicts and **r**edundant authorisation policies. For each event description $\mathcal{E}^i_x$, our compiler generated an optimised event description $\mathcal{E}^o_x$ by transforming the translatable simple FVP definitions in $\mathcal{E}^i_x$ into statically determined FVP definitions. In all cases, the compilation time did not exceed half a second. Figure 1 (top) shows the reduction in the number of rules and conditions in the compiled event descriptions. Compiling $\mathcal{E}^i_h$, e.g.,

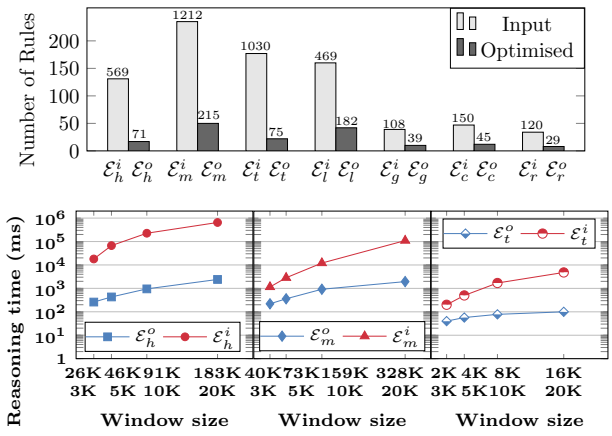

Figure 1: Event description optimisation (top figure); the number above each bar denotes the total number of rule conditions. Reasoning over input and compiled/optimised event descriptions (bottom figure); the horizontal axes denote the avg number of input events (top) and the avg number of computed FVP intervals (bottom) per window.

which includes *131* rules and *569* rule conditions in total, leads to $\mathcal{E}^o_m$, containing *17* rules, with a total of *71* conditions. Our results show that our compiler can transform event descriptions for real applications into a significantly more compact form, paving the way for code maintenance.

We employed RTEC for CER, with event descriptions $\mathcal{E}^i_h$, $\mathcal{E}^o_h$, $\mathcal{E}^i_m$, $\mathcal{E}^o_m$, $\mathcal{E}^i_t$ and $\mathcal{E}^o_t$. We could not run RTEC on the remaining event descriptions because no datasets for the corresponding applications were available to us. RTEC processes input streams using a sliding window. Figure 1 (bottom) shows the reasoning times of RTEC per window, as the window size increased. Each data point is the average of 30 queries. In all of our experiments, reasoning with an optimised event description led to the same FVP intervals as reasoning with the original event description. Our results demonstrate that reasoning with an optimised event description may lead to efficiency benefits of about 1-2 orders of magnitude. Due to the significant size reduction we observed when compiling $\mathcal{E}^i_l$, $\mathcal{E}^i_g$, $\mathcal{E}^i_c$ and $\mathcal{E}^i_r$ (see Figure 1 (top)), we expect that their optimisation will lead to comparable reasoning efficiency benefits with the ones of Figure 1 (bottom).

Summary

The Event Calculus is a formal computational framework used in a wide range of tasks, including CER, legal contract verification, clinical guideline monitoring and distributed authorisation policy management. Unfortunately, the knowledge engineer does not always employ the optimal Event Calculus representation. To address this issue, we identified the class of simple FVP definitions that can be translated into statically determined FVP definitions. We provided a compiler for the translation, and a reproducible empirical evaluation on real applications. Our compiler reduced the size of the event descriptions by orders of magnitude, leading to significant reasoning efficiency gains.

[2]https://tinyurl.com/caviardataset

[3]https://zenodo.org/record/1167595

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

Acknowledgements

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

References

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

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

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

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

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

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

Bottrighi, A.; Chesani, F.; Mello, P.; Montali, M.; Montani, S.; and Terenziani, P. 2011. Conformance Checking of Executed Clinical Guidelines in Presence of Basic Medical Knowledge. In *Business Process Management Workshops*, volume 100 of *Lecture Notes in Business Information Processing*, 200–211. Springer.

Bragaglia, S.; Chesani, F.; Mello, P.; Montali, M.; and 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*, volume 7360, 123–146.

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

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

Chittaro, L.; and Montanari, A. 1996. Efficient Temporal Reasoning in the Cached Event Calculus. *Comput. Intell.*, 12(3): 359–382.

Clark, K. L. 1977. Negation as Failure. In *Logic and Data Bases*, 293–322. Plemum Press.

Eiter, T.; Ogris, P.; and Schekotihin, K. 2019. A Distributed Approach to LARS Stream Reasoning (System paper). *Theory Pract. Log. Program.*, 19(5-6): 974–989.

Falcionelli, N.; Sernani, P.; de la Torre, A. B.; Mekuria, D. N.; Calvaresi, D.; Schumacher, M.; Dragoni, A. F.; and Bromuri, S. 2019. Indexing the Event Calculus: Towards practical human-readable Personal Health Systems. *Artif. Intell. Medicine*, 96: 154–166.

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

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

Kowalski, R.; and Sergot, M. 1986. A Logic-Based Calculus of Events. *New Gen. Computing*, 4(1): 67–96.

Mantenoglou, P.; Kelesis, D.; and Artikis, A. 2023. Complex Event Recognition with Allen Relations. In *KR*, 502–511.

Mantenoglou, P.; Pitsikalis, M.; and Artikis, A. 2022. Stream Reasoning with Cycles. In *KR*, 544–553.

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

Parvizimosaed, A.; Sharifi, S.; Amyot, D.; Logrippo, L.; Roveri, M.; Rasti, A.; Roudak, A.; and Mylopoulos, J. 2022. Specification and analysis of legal contracts with Symboleo. *Softw. Syst. Model.*, 21(6): 2395–2427.

Paschke, A.; and Bichler, M. 2008. Knowledge Representation Concepts for Automated SLA Management. *Decision Support Systems*, 46(1): 187–205.

Pitsikalis, M.; Artikis, A.; Dreo, R.; Ray, C.; Camossi, E.; and Jousselme, A. 2019. Composite Event Recognition for Maritime Monitoring. In *DEBS*, 163–174.

Sharifi, S.; Parvizimosaed, A.; Amyot, D.; Logrippo, L.; and Mylopoulos, J. 2020. Symboleo: Towards a Specification Language for Legal Contracts. In *IEEE RE*, 364–369.

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

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

Zahoor, E.; Chaudhary, M.; Akhtar, S.; and Perrin, O. 2023. A formal approach for the identification of redundant authorization policies in Kubernetes. *Comput. Secur.*, 135: 103473.

Zahoor, E.; Ikram, A.; Akhtar, S.; and Perrin, O. 2022. A Formal Approach for the Identification of Authorization Policy Conflicts within Multi-Cloud Environments. *J. Grid Comput.*, 20(2): 18.